

SEKE

2019

**Proceedings of the 31st
International Conference on
Software Engineering &
Knowledge Engineering**

**Lisbon, Portugal
July 10-12, 2019**



PROCEEDINGS
SEKE 2019

**The 31st International Conference on
Software Engineering &
Knowledge Engineering**

Sponsored by

KSI Research Inc. and Knowledge Systems Institute Graduate School, USA

Technical Program

July 10 – 12, 2019

Hotel Tivoli, Lisbon, Portugal

Organized by

KSI Research Inc. and Knowledge Systems Institute Graduate School, USA

Copyright © 2019 by KSI Research Inc. and Knowledge Systems Institute Graduate School

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher.

ISBN: 1-891706-48-9

ISSN: 2325-9000 (print)

2325-9086 (online)

DOI reference number: 10.18293/SEKE2019

Publisher Information:

KSI Research Inc. and Knowledge Systems Institute Graduate School

156 Park Square

Pittsburgh, PA 15238 USA

Tel: +1-412-606-5022

Fax: +1-847-679-3166

Email: seke@ksiresearch.org

Web: <http://ksiresearchorg.ipage.com/seke/seke19.html>

Proceedings preparation, editing and printing are sponsored by KSI Research Inc. and Knowledge Systems Institute Graduate School, USA.

Printed by KSI Research Inc. and Knowledge Systems Institute Graduate School

FOREWORD

Welcome to the 31st International Conference on Software Engineering and Knowledge Engineering (SEKE), in Hotel Tivoli, Lisbon, Portugal. In last 30 years, SEKE has established itself as a major international forum to foster, among academia, industry, and government agencies, discussion and exchange of ideas, research results and experience in software engineering and knowledge engineering. The SEKE community has grown to become a very important and influential source of ideas and innovations on the interplays between software engineering and knowledge engineering, and its impact on the knowledge economy has been felt worldwide. On behalf of the Program Committee, it is my great pleasure to invite you to participate, not only in the technical program of SEKE 2019 and its rich assortment of activities, but also in enjoying the stunning capital of Portugal and one of the great capitals of Western Europe. Lisbon is a city with more than 800 years where historical heritage, modernism, culture and nightlife combine in a perfect harmony. The excellence of the climate and the sympathy of the Portuguese population associated with the diversity and quality of the opportunities offered leave visitors with a desire to return to Lisbon. Lisbon and Portugal have recently won several international awards as destinations of excellence, such as those from the World Travel Award.

This year, we received 225 submissions from 29 countries. Through a rigorous review process where a majority of the submitted papers received three reviews, and the rest with two reviews, we were able to select 88 full papers for the general conference (39.1 percent), 61 short papers (27.1 percent), 3 posters (1.4 percent), 1 demo (0.4 percent) and 72 rejects (32 percent). Out of that, 9 papers have been specifically submitted, and later accepted, for the 4 special sessions (Theoretical Software Engineering TSE 4 papers, Semantic Enabled Software Engineering SESE 1 paper, Knowledge Graphs KG 3 papers and Machine Learning for SE and KE MLA 1 paper), and 140 papers are scheduled for presentation in forty sessions during the conference. In addition, the technical program includes two excellent keynote speeches from Professor Robert Laurini and Professor Rui L. Aguiar.

The high quality of the SEKE 2019 technical program would not have been possible without the tireless effort and hard work of many individuals. First of all, we would like to express our sincere appreciation to all the authors whose technical contributions have made the final technical program possible. We are very grateful to all the Program Committee members whose expertise and dedication made our responsibility that much easier. Our gratitude also goes to the keynote speakers who graciously agreed to share their insight on important research issues, to the conference organizing committee members for their superb work, and to the external reviewers for their contribution.

Personally, we owe a debt of gratitude to a number of people whose help and support with the technical program and the conference organization are unfailing and indispensable. We are deeply indebted to Dr. S. K. Chang, Chair of the Steering Committee, for his constant guidance and support that are essential to pull off SEKE 2019. Our heartfelt appreciation goes to Dr. Oscar Mortagua Pereira, University of Aveiro, Portugal, the Conference Chair, for his help and experience. In addition, we also like to express our appreciation to Prof. Jing Sun, The University of Auckland, New Zealand, to Prof. Iakov Exman, The Jerusalem College of Engineering, Israel, to Dr. Yucong Duan, Hainan University, China and to Dr. Gunasekaran Manogaran, University of California, Davis, USA for their excellent job in organizing the special sessions SESE, TSE, KG and MLA, respectively.

We would like also to express our great appreciation to all of the conference organization committee members, including the Publicity Chair, Lan Lin, Ball State University, USA and Nuno Antunes, University of Coimbra, Portugal. Moreover, we would like to appreciate and recognize our Conference Liaisons in different regions for their important contributions. They are: Asia Liaison – Hironori Washizaki, Waseda University, Japan; Australasia Liaison – Jing Sun, The University of Auckland, New Zealand; Europe Liaison - Raul Garcia Castro, Universidad Politecnica de Madrid, Spain; India Liaison - Swapan Bhattacharya, National Institute of Technology Karnataka, Surathakl; and South America Liaison - Jose Carlos Maldonado, ICMC-USP, Brazil.

Last but certainly not the least, we must acknowledge the important contributions that the KSI staff members have made. Their timely and dependable support and assistance throughout the entire process have been truly remarkable. Finally, we wish you have productive discussion, great networking, effective presentation, and pleasant stay and travel in Lisbon, Portugal to participate in SEKE 2019.

Angelo Perkusich, Federal University of Campina Grande, Brazil, Program Committee Chair
Raul Garcia Castro, Universidad Politecnica de Madrid, Spain, Program Committee Co-Chair
Diogo Regateiro, Institute de Telecomunicacoes, Portugal, Program Committee Co-Chair

SEKE 2019

The 31st International Conference on Software Engineering & Knowledge Engineering

July 10 – 12, 2019

Hotel Tivoli, Lisbon, Portugal

Conference Organization

CONFERENCE CHAIR

Oscar Mortagua Pereira, University of Aveiro, Portugal

PROGRAM COMMITTEE CHAIR

Angelo Perkusich, Federal University of Campina Grande, Brazil

PROGRAM COMMITTEE CO-CHAIRS

Raul Garcia Castro, Universidad Politecnica de Madrid, Spain

Diogo Regateiro, Institute de Telecomunicacoes, Portugal

STEERING COMMITTEE CHAIR

Shi-Kuo Chang, University of Pittsburgh, USA

STEERING COMMITTEE

Vic Basili, University of Maryland, USA

Bruce Buchanan, University of Pittsburgh, USA

C. V. Ramamoorthy, University of California, Berkeley, USA

ADVISORY COMMITTEE

Jerry Gao, San Jose State University, USA

Swapna Gokhale, University of Connecticut, USA

Xudong He, Florida International University, USA

Natalia Juristo, Universidad Politecnica de Madrid, Spain

Taghi Khoshgoftaar, Florida Atlantic University, USA

Guenther Ruhe, University of Calgary, Canada

Masoud Sadjadi, Florida International University, USA

Du Zhang, California State University, USA

PROGRAM COMMITTEE

Silvia Teresita Acuna, Universidad Autonoma de Madrid, Spain
Shadi Alawneh, Oakland University, USA
Vaibhav Anu, North Dakota State University, USA
Doo-Hwan Bae, Korea Advanced Institute of Science and Technology, Korea
Fevzi Belli, University of Paderborn, Germany
Ateet Bhalla, Consultant, India
Swapan Bhattacharya, NITK, Surathakl, India
Ivo Bukovsky, Czech Technical University in Prague, Czech Republic
Guoray Cai, Penn State University, USA
Raul Garcia Castro, Universidad Politecnica de Madrid, Spain
Keith Chan, Hong Kong Polytechnic University, Hong Kong
Wen-Hui Chen, National Taipei University of Technology, Taiwan
Maria Francesca Costabile, University of Bari, Italy
Lin Deng, Towson University, USA
Derek Doran, Wright State University, USA
Weichang Du, University of New Brunswick, Canada
Christof Ebert, Vector Consulting Services, Germany
Magdalini Eirinaki, San Jose State University, USA
Abdelrahman Osman Elfaki, University of Tabuk, Saudi Arabia
Ruby ElKharboutly, Quinnipiac University, Canada
Honghao Gao, ShangHai University, China
Kehan Gao, Eastern Connecticut State University, USA
Felix Garcia, University of Castilla-La Mancha, Spain
Olivier Le Goaer, University of Pau, France
Swapna Gokhale, Univ. of Connecticut, USA
Wolfgang Golubski, Zwickau University of Applied Sciences, Germany
Hassan Haghighi, Shahid Beheshti University, Iran
Hao Han, National Institute of Informatics, Japan
Xudong He, Florida International University, USA
Shihong Huang, Florida Atlantic University, USA
Hamdy Ibrahim, University of Calgary, Canada
Bassey Isong, North-West University, South Africa
Clinton Jeffery, University of Idaho, USA
Jason Jung, Chung-Ang University, South Korea
Pankaj Kamthan, Concordia University, Canada
Ananya Kanjilal, B.P. Poddar Institute of Technology and Management, India
Taghi Khoshgoftaar, Florida Atlantic University, USA
Jun Kong, North Dakota State University, USA
Aneesh Krishna, Curtin University of Technology, Australia
Vinay Kulkarni, Tata Consultancy Services, India
Bixin Li, Southeast University, China
Yuan-Fang Li, Monash University, Australia
Jianhua Lin, Eastern Connecticut State University, USA
Lan Lin, Ball State University, USA
Xiaodong Liu, Edinburgh Napier University, United Kingdom
Luanna Lopes Lobato, Federal University of Goias, Brazil
Baojun Ma, Beijing University of Posts and Telecommunications, China
Beatriz Marin, Universidad Diego Portales, Chile
Riccardo Martoglia, University of Modena and Reggio Emilia, Italy
Santiago Matalonga, University of the West of Scotland, UK
Andre Menolli, Universidade Estadual do Norte do Parana (UENP), Brazil
Hiroyuki Nakagawa, Osaka University, Japan
Alex Norta, Tallinn University of Technology, Estonia
Edson A. Oliveira Jr., State University of Maringa, Brazil
Oscar Mortagua Pereira, University of Aveiro, Portugal

Antonio Piccinno, University of Bari, Italy
 Alfonso Pierantonio, University of L'Aquila, Italy
 Rick Rabiser, Johannes Kepler University, Austria
 Claudia Raibulet, University of Milan, Italy
 Damith C. Rajapakse, National University of Singapore, Singapore
 Rajeev Raje, IUPUI, USA
 Henrique Rebelo, Universidade Federal de Pernambuco, Brazil
 Marek Reformat, University of Alberta, Canada
 Diogo Regateiro, Institute de Telecomunicacoes, Portugal
 Stephan Reiff-Marganiec, Leicester University, United Kingdom
 Daniel Rodriguez, Universidad de Alcala, Spain
 Masoud Sadjadi, Florida International University, USA
 Claudio Sant'Anna, Universidade Federal da Bahia, Brazil
 Klaus-Dieter Schewe, SCCH, Austria
 Abdelhak-Djamel Seriai, University of Montpellier 2 for Sciences and Technology, France
 Michael Shin, Texas Tech University, USA
 Vijayan Sugumaran, Oakland University, USA
 Jing Sun, University of Auckland, New Zealand
 Meng Sun, Peking University, China
 Yanchun Sun, Peking University, China
 Gerson Sunye, University of Nantes, France
 Kumiko Tadano, NEC, Japan
 Chuanqi Tao, Nanjing University of Science and Technology, China
 Mark Trakhtenbrot, Holon Institute of Technology, Israel
 Peter Troeger, TU Chemnitz, Germany
 Christelle Urtado, LGI2P Ecole des Mines d'Ales, France
 Sylvain Vauttier, Ecole des mines d'Ales, France
 Silvia Vergilio, Federal University of Parana (UFPR), Brazil
 Gennaro Vessio, University of Bari, Italy
 Sergiy Vilkomir, East Carolina University, USA
 Aaron Visaggio, University of Sannio, Italy
 Ye Wang, Zhejiang Gongshang University, China
 Yong Wang, New Mexico Highlands University, USA
 Zhongjie Wang, Harbin Institute of Technology, China
 Hironori Washizaki, Waseda University, Japan
 Bingyang Wei, Midwestern State University, USA
 Guido Wirtz, Bamberg University, Germany
 Franz Wotawa, TU Graz, Austria
 Peng Wu, Institute of Software, Chinese Academy of Sciences, China
 Qing Wu, Hangzhou Dianzi University, China
 Dianxiang Xu, Boise State University, USA
 Frank Weifeng Xu, University of Baltimore, USA
 Haiping Xu, University of Massachusetts Dartmouth, USA
 Lai Xu, Bournemouth University, UK
 Guowei Yang, Texas State University, USA
 Yuyu Yin, Hangzhou Dianzi University, China
 Huiqun Yu, East China University of Science and Technology, China
 Du Zhang, Macau University of Science and Technology, China
 Pengcheng Zhang, Hohai University, China
 Yong Zhang, Tsinghua University, China
 Zhenyu Zhang, Institute of Software, Chinese Academy of Sciences, China
 Zhigao Zheng, Central China Normal University, USA
 Nianjun Zhou, IBM, USA
 Huibiao Zhu, East China Normal University, China
 Eugenio Zimeo, University of Sannio, Italy

PUBLICITY CHAIR

Lan Lin, Ball State University, USA
Nuno Antunes, University of Coimbra, Portugal

ASIA LIAISON

Hironori Washizaki, Waseda University, Japan

AUSTRALASIA LIAISON

Jing Sun, The University of Auckland, New Zealand

EUROPE LIAISON

Raul Garcia Castro, Universidad Politecnica de Madrid, Spain

INDIA LIAISON

Swapn Bhattacharya, National Institute of Technology Karnataka, Surathakl, India

SOUTH AMERICA LIAISON

Jose Carlos Maldonado, ICMC-USP, Brazil

Keynote I

Introduction to Mobile Applications for Smart Cities

Professor Robert Laurini
Professor Emeritus
University of Lyon
Lyon, France

Abstract

Smart Cities can be defined with a digital layer covering several aspects of urban governance, based on four pillars such as knowledge engineering, deep learning, realtime massive data and geovisualization. After explaining the issues relative to those pillars, we will explain how they can be combined to support and develop Location-Based Services and Mobile Applications. In this talk, three categories of applications will be outlined, those based on mobility, those for which the answer varies from the location of the mover (space), and continuous applications for which also time is important.

About the Speaker

Professor Robert Laurini holds two doctorates (1973 and 1980) from the University of Lyon, France in which he was professor in information technologies at INSA-Lyon. He is a specialist in geographic information systems and more recently on knowledge engineering for smart cities. In 76-77, he was researcher at the Martin Centre, University of Cambridge, UK; in 86-87, visiting professor at the University of Maryland, College Park; during the period 1990-2000, he was part-time professor at the IUAV University of Venice, Italy. He has supervised or co-supervised 45 PhD's. He was involved in PhD committees in 17 countries. He has authored or co-authored more than 250 papers and 8 books, the last of them is "Geographic Knowledge Infrastructure, Applications to Territorial Intelligence and Smart Cities". He was one of the founders of the ACM Spatial Group, European editor of "Computers, Environment and Urban Systems" and associated editor of the "Journal for Visual Languages and Computing". He is now fellow of the Knowledge Systems Institute, USA. He fluently speaks English, French, Italian and Spanish. He is also the founder of "Academics Without Borders", which is an NGO devoted to modernizing universities in developing countries. For more details please see: <http://www.laurini.net/robert/>

Keynote II

5G - After the initial launch, is all about software

Professor Rui L. Aguiar
Institute of Telecommunications/DETI
University of Aveiro
Aveiro, Portugal

Abstract

The talk will briefly highlight the changes to be brought by the upcoming 5G networks, and present a view of the challenges and conceptual changes that the new wireless networks will bring in general, and to software engineering and knowledge engineering in particular. The discussion will touch on the control aspects of the networks, the requirements on novel control abstractions, and on new service development concepts and interfaces, focusing on the new virtualized and softwarized networks that will arise with 5G.

About the Speaker

Rui L. Aguiar received his Ph.D. degree in 2001 from the University of Aveiro. He is currently a Full Professor at the University of Aveiro, responsible for the networking area, and has been previously an adjunct professor at the INI, Carnegie Mellon University. He is coordinating a research line nationwide in Instituto de Telecomunicações, on the area of Networks and Multimedia. His current research interests are centred on the implementation of advanced networks and systems, with special emphasis on 5G networks and the Future Internet. He has about five hundred published papers in those areas. He has served as technical and general chair of several conferences, from IEEE, ACM and IFIP. He sits on the TPC of all major IEEE ComSoc conferences. He has extensive participation in national and international projects, and in industry technology transfer actions. He is the current Chair of the steering board of the Networld2020 ETP. He is senior member of IEEE, Portugal ComSoc Chapter Chair, and a member of ACM. He is associated editor of several journals and sits on the Advisory Board of several EU-projects and research units from several countries.

Table of Contents

Session Knowledge Graphs	
A DIK-based Question-Answering Architecture with Multi-sources Data for Medical Self-Service (S).....	1
<i>Menglong Li, Mengxing Huang, Yu Zhang and Wenlong Feng</i>	
Constructing a Knowledge Base of Coding Conventions from Online Resources	5
<i>Junming Cao, Tianjiao Du, Beijun Shen, Wei Li, Qingyue Wu and Yuting Chen</i>	
Combining time, keywords and authors information to construct papers correlation graph (S)	11
<i>Hanwen Liu, Huaizhen Kou, Xiaoxiao Chi and Lianyong Qi</i>	
Modeling and Simulation of CPS based on SysML and Modelica (S)	15
<i>Fei Deng, Yunqiang Yan, Feng Gao and Linbo Wu</i>	
Session Agile Software I	
AgileCritPath: Identifying Critical Tasks in Agile Environments.....	20
<i>Rachel Vital, Glaucia Melo dos Santos, Toacy Oliveira, Paulo Alencar and Don Cowan</i>	
Evaluating Software Developers' Acceptance of a Tool for Supporting Agile Non-Functional Requirement Elicitation.....	26
<i>Felipe Ramos, Antonio Pedro, Marcos Cesar, Alexandre Costa, Mirko Perkusich, Hyggo Almeida and Angelo Perkusich</i>	
Automatic Generation of Virtual Assistants from Databases using Active Ontologies.....	32
<i>Martin Blersch, Sebastian Weigelt, Walter Tichy and Kevin Angele</i>	
GADIS: A Genetic Algorithm for Database Index Selection (S)	39
<i>Priscilla Neuhaus, Julia Couto, Jonatas Wehrmann, Duncan Ruiz and Felipe Meneguzzi</i>	
Session Software Testing I	
Research on Page Object Generation Approach for Web Application Testing.....	43
<i>Yimei Chen, Zheng Li, Ruilian Zhao and Junxia Guo</i>	
A Class-level Test Selection Approach Toward Full Coverage For Continuous Integration..	49
<i>Yingling Li, Junjie Wang, Qing Wang and Jun Hu</i>	
Amplifying Tests for Cross-Platform Apps through Test Patterns.....	55
<i>Thiago Botti de Assis, André Augusto Menegassi and André Takeshi Endo</i>	
Session Theoretical Software Engineering	
Algebraic Convergence to Software-Knowledge: Deep Software Learning (P)	61
<i>Iaakov Erman and Assaf Spanier</i>	
Formal ontologies and data shapes within the Software Engineering development lifecycle.	64
<i>Jose María Alvarez Rodríguez, Valentín Moreno and Juan Llorens</i>	

Towards an Ontology to Support Decision-making in Hospital Bed Allocation (S).....	71
<i>Debora Engelmann, Julia Couto, Vagner Gabriel, Renata Vieira and Rafael Bordini</i>	
A Software System is Greater than its Modules' Sum: Providers & Consumers' Modularity Matrix.....	75
<i>Iaakov Exman and Harel Wallach</i>	

Session Agile Software

An Effort Estimation Support Tool for Agile Software Development: An Empirical Evaluation.....	82
<i>Emanuel Dantas, Alexandre Costa, Marcus Vinicius, Mirko Perkusich, Hyggo Almeida and Angelo Perkusich</i>	
Towards an artifact to support agile teams in software analytics activities.....	88
<i>Joelma Choma, Eduardo Guerra, Tiago Silva Da Silva, Luciana Zaina and Filipe Figueiredo Correia</i>	
Sprint Performance Forecasts in Agile Software Development - The Effect of Futurespectives on Team-Driven Dynamics.....	94
<i>Fabian Kortum, Jil Klünder, Wasja Brunotte and Kurt Schneider</i>	

Session Software Testing II

Research on Multi-constraint Combinatorial Test Technology for High Confidence Embedded Software (S).....	102
<i>Feng Gao, Fei Deng and Yunqiang Yan</i>	
Specification-based Testing with Simulation Relations (S).....	107
<i>Canh Minh Do and Kazuhiro Ogata</i>	
A Survey Study on the Inference Problem in Distributed Environment (S).....	113
<i>Adel Jebali, Abderrazak Jemai and Salma Sassi</i>	
Towards human-centric software testing.....	117
<i>Samantha Catania, Chris Porter and Mark Micallef</i>	

Session Software Testing III

Semantic Analysis for Deep Q-Network in Android GUI Testing.....	123
<i>Thi Anh Tuyet Vuong and Shingo Takada</i>	
Impacts of Data Uniformity in the Reuse of Acceptance Test Glue Code.....	129
<i>Douglas Hiura Longo, Patrícia Vilain and Lucas Pereira da Silva</i>	
Test Case Generation by EFSM Extracted from UML Sequence Diagrams.....	135
<i>Mauricio Rocha, Adenilson Simão, Thiago Sousa and Marcelo Batista</i>	
The Smell of Blood: Evaluating Anemia and Bloodshot Symptoms in Web Applications..	141
<i>Zijie Huang, Junhua Chen and Jianhua Gao</i>	

Session Formal Methods I

Formalization and Verification of RTPS StatefulWriter Module Using CSP	147
<i>Jiaqi Yin, Huibiao Zhu, Yuan Fei, Qiwen Xu and Ruobiao Wu</i>	
A Sound and Complete Axiomatisation for Spatio-Temporal Specification Language	153
<i>Tengfei Li, Jing Liu, Dongdong An and Haiying Sun</i>	
Formal Specification and Model Checking of the Lim-Jeong-Park-Lee Autonomous Vehicle Intersection Control Protocol (S)	159
<i>Moe Nandi Aung, Yati Phyo and Kazuhiro Ogata</i>	
<hr/> Session Bayesian Methods <hr/>	
Improving the Applicability of the Ranked Nodes Method to build Expert-Driven Bayesian Networks (S)	165
<i>João Nunes, Luiz Silva, Mirko Perkusich, Kyller Gorgonio, Hyggo Almeida and Angelo Perkusich</i>	
A systematic process to define expert-driven software metrics thresholds (S)	171
<i>Renata Saraiva, Mirko Perkusich, Hyggo Almeida and Angelo Perkusich</i>	
Multi-source fault detection and diagnosis based on multi-level Knowledge Graph and Bayesian theory reasoning (S)	177
<i>Tao Sun and Qi Wang</i>	
<hr/> Session Formal Methods II <hr/>	
Formal Specification and Model Checking of A* Algorithm	181
<i>Kazuhiro Ogata</i>	
PAT approach to Architecture Behavioural Verification	187
<i>Nacha Chondamrongkul, Jing Sun and Ian Warren</i>	
Leveraging Rigorous Software Specification Towards Systematic Detection of SDN Control Conflicts (S)	193
<i>Xin Sun and Lan Lin</i>	
<hr/> Session Software Bugs <hr/>	
The Influence of God Class and Long Method in the Occurrence of Bugs in Two Open Source Software Projects: An Exploratory Study (S)	199
<i>Aloisio Cairo, Glauco Carneiro, Antonio Resende and Fernando Brito E Abreu</i>	
Feature Evaluation for Automatic Bug Report Summarization (S)	205
<i>Akalanka Galappaththi and John Anvik</i>	
Generating Integration Tests Automatically Using Frequent Patterns of Method Execution Sequences	209
<i>Mark Grechanik and Gurudev Devanla</i>	
CrashAwareDev: Supporting Software Development based on Crash Report Mining and Analysis (S)	215
<i>Leandro Beserra and Roberta Coelho</i>	
<hr/> Session Internet of Things <hr/>	

Dynamic and Interoperable Control of IoT Devices and Applications based on Calvin Framework	221
<i>Fernanda Famá, Cleuves de Carvalho, Danilo Santos, Angelo Perkusich and Kyller Gorgônio</i>	
Reverse Engineering Behavioural Models of IoT Devices	227
<i>Sébastien Salva and Elliott Blot</i>	
A Resource Management Architecture For Exposing Devices as a Service in the Internet of Things	233
<i>Carlos Pantoja, Heder Dorneles Soares, Tielle Alexandre, Jose Viterbo and Amal El-Fallah Seghrouchni</i>	
<hr/> Session Security <hr/>	
SPRO: Security Process Framework	239
<i>Henrique Persch, Lisandra Fontoura and Adriano Fontoura</i>	
Detecting Security Vulnerabilities using Clone Detection and Community Knowledge	245
<i>Fabien Patrick Viertel, Wasja Brunotte, Daniel Strüder and Kurt Schneider</i>	
Case-Based Cybersecurity Incident Resolution	253
<i>Marcelo Colomé, Raul Ceretta Nunes and Luis Alvaro de Lima Silva</i>	
<hr/> Session Formal Methods III <hr/>	
Verifying Static Aspects of UML models using Prolog (S)	259
<i>Feng Sheng, Huibiao Zhu, Zongyuan Yang, Jiaqi Yin and Gang Lu</i>	
Modeling and Verifying TESAC Using CSP	265
<i>Dongzhen Sun, Huibiao Zhu, Yuan Fei, Lili Xiao, Gang Lu and Jiaqi Yin</i>	
PRISM Code Generation for Verification of Mediator Models (S)	271
<i>Weidi Sun and Meng Sun</i>	
<hr/> Session Adaptive Software <hr/>	
An evolutionary model for dynamic and adaptative service composition in distributed environment	275
<i>Jiawei Lu, Huan Zhou, Jun Xu, Haibo Pan and Gang Xiao</i>	
Learning - based Adaptation Framework for Elastic Software Systems	281
<i>Yingcheng Sun, Xiaoshu Cai and Kenneth Loparo</i>	
Self-Adaptive software changes analysis method based on "Detection-Recognition" Mechanism (S)	287
<i>He Zhang, Qingshan Li, Lu Wang and Wen Cheng</i>	
<hr/> Session Semantic Enabled SE <hr/>	
morph-GraphQL: GraphQL Servers Generation from R2RML Mappings (S)	291
<i>Freddy Priyatna, David Chaves-Fraga, Ahmad Alobaid and Oscar Corcho</i>	

Semantic Rule Based Program Monitoring (S).....	297
<i>Luke Tudor, Jing Sun, Hai H. Wang and Bingyang Wei</i>	
Context-aware Reactive Systems based on Runtime Semantic Models (S).....	301
<i>Ester Giallonardo, Francesco Poggi, Davide Rossi and Eugenio Zimeo</i>	
Enhancing Semantic Search of Crowdsourcing IT Services using Knowledge Graph.....	307
<i>Duankang Fu, Shufan Zhou, Beijun Shen and Yuting Chen</i>	
<hr/> Session Empirical Studies <hr/>	
An Empirical Study on Research and Developmental Opportunities in Refactoring Practices.....	313
<i>Shivani Jain and Anju Saha</i>	
An Empirical Studies on Optimal Solutions Selection Strategies for Effort-Aware Just-in-Time Software Defect Prediction.....	319
<i>Xingguang Yang, Huiqun Yu, Guisheng Fan and Kang Yang</i>	
Empirical Studies Concerning the Maintenance of BPMN Diagrams: A Systematic Mapping Study.....	325
<i>Ursula Campos, Adriana Lopes, Simone Barbosa and Tayana Conte</i>	
<hr/> Session Neural Networks <hr/>	
Multistep Flow Prediction on Car-Sharing Systems: A Multi-Graph Convolutional Neural Network with Attention Mechanism.....	331
<i>Yi Luo, Qin Liu, Hongming Zhu, Hongfei Fan, Tianyou Song, Chang Wu Yu and Bowen Du</i>	
Chinese Text Relation Extraction with Multi-instance Multi-label BLSTM Neural Networks.....	337
<i>Liubo Ouyang, Hui Tang and Guangyi Xiao</i>	
A Convolutional Neural Network Pruning Method Based On Attention Mechanism.....	343
<i>Xiao Jie Wang, Wenbin Yao and Huiyuan Fu</i>	
An Integrated Software Vulnerability Discovery Model based on Artificial Neural Network	349
<i>Gul Jabeen, Luo Ping, Junaid Akram and Akber Aman Shah</i>	
<hr/> Session Machine Learning Algorithms for SE <hr/>	
Machine Learning for Learnability of MDD tools.....	355
<i>Saad Bin Abid, Vishal Mahajan and Levi Lucio</i>	
Assessing the Influence of Size Category of the Project in God Class Detection, an Experimental Approach based on Machine Learning.....	361
<i>Khalid Alkharabsheh, Yania Crespo, Manuel Fernandez-Delgado, José M. Cotos and Jose Angel Taboada</i>	
A Services Development Approach for Smart Home Based on Natural Language Instructions.....	367
<i>Yiyang Chen, Zhanghui Liu, Zhiming Huang, Chuangshumin Hu and Xing Chen</i>	

Modeling User Contextual Behavior Semantics with Geographical Influence for Point-Of-Interest Recommendation.....	373
<i>Dongjin Yu, Kaihui Xu and Dongjing Wang</i>	

Session Deep Learning/AI

A Deep Learning Model Based on Sparse Matrix for Point-of-Interest Recommendation...	379
<i>Jun Zeng, Haoran Tang, Yinghua Li and Xin He</i>	
Improving Code Generation From Descriptive Text By Combining Deep Learning and Syntax Rules.....	385
<i>Xiangru Tang, Zhihao Wang, Jiyang Qi and Zengyang Li</i>	
Safe-by-Design Development Method for Artificial Intelligent Based Systems.....	391
<i>Gabriel Pedroza and Adedjouma Morayo</i>	

Session Software Development II

Augmenting App Review with App Changelogs: An Approach for App Review Classification.....	398
<i>Chong Wang, Tao Wang, Peng Liang, Maya Daneva and Marten Sinderen</i>	
Prudent Practices for Designing Virtual Desktop Experiments.....	404
<i>Peiyu Liu, Wenzhi Chen, Zonghui Wang and Lirong Fu</i>	
CrowDevBot: A Task-Oriented Conversational Bot for Software Crowdsourcing Platform (S).....	410
<i>Zeyu Ni, Beijun Shen, Yuting Chen, Zhangyuan Meng and Junming Cao</i>	
Retrieving Curated Stack Overflow Posts from Project Task Similarities (S).....	415
<i>Glaucia Santos, Toacy Oliveira, Paulo Alencar and Don Cowan</i>	

Session Software Defects

Software Defect Prediction Model Based on Improved Deep Forest and AutoEncoder by Forest.....	419
<i>Wenbo Zheng, Shaocong Mo, Xin Jin, Yili Qu, Zefeng Xie and Jia Shuai</i>	
Multi-project Regression based Approach for Software Defect Number Prediction.....	425
<i>Qiguo Huang, Chao Ni, Xiang Chen, Qing Gu and Kaibo Cao</i>	
Cross-Project Defect Prediction via Transferable Deep Learning-Generated and Handcrafted Features.....	431
<i>Shaojian Qiu, Lu Lu, Ziyi Cai and Siyu Jiang</i>	
An Investigation of Ensemble Approaches to Cross-Version Defect Prediction.....	437
<i>Xiaoxing Yang, Xin Li, Wushao Wen and Jianmin Su</i>	

Session Software Architectures I

A Multilevel Analysis Method for Architecture Erosion.....	443
<i>Tong Wang, Dongdong Wang and Bixin Li</i>	

The affinity Platform: Modular Architecture based on Independent Components (S)	449
<i>Alexandru Ardelean and Kuderna-Iulian Bența</i>	

A Mapping Study about Data Lakes: An Improved Definition and Possible Architectures .	453
<i>Julia Couto, Olimar Borges, Duncan D. Ruiz, Sabrina Marczak and Rafael Prikladnicki</i>	

Session Software Architectures II

Architecture for Discovery and Customization of Multi-tenant Learning Process as a service and resources allocation in cloud computing.	459
<i>Sameh Azouzi, Zaki Brahmi and Sonia Ayachi Ghannouchi</i>	

An Empirical Study about Software Architecture Configuration Practices with the Java Spring Framework (S)	465
<i>Quentin Perez, Alexandre Le Borgne, Christelle Urtado and Sylvain Vauttier</i>	

Recover and Optimize Software Architecture Based on Source Code and Directory Hierarchies (S)	469
<i>Tong Wang, Yelian Zhang, Xufang Gong and Bixin Li</i>	

Session User Centered Design

Automated user-oriented description of emerging composite ambient applications	473
<i>Maroun Koussaifi, Sylvie Trouilhet, Jean-Paul Arcangeli and Jean-Michel Bruel</i>	

Usability of Chatbots: A Systematic Mapping Study	479
<i>Ranci Ren, John W. Castro, Silvia T. Acuna and Juan de Lara</i>	

Extending Behavior-Driven Development for Assessing User Interface Design Artifacts (S)	485
<i>Thiago Silva, Marco Winckler and Hallvard Trætteberg</i>	

Session Software Development III

A Method to Recommend Artifacts to New Tasks in Software Projects (S)	489
<i>Edson Lucas, Toacy Oliveira and Paulo Alencar</i>	

SOTagger - Towards Classifying Stack Overflow Posts through Contextual Tagging (S) . . .	493
<i>Akhila Sri Manasa Venigalla, Chaitanya S. Lakkundi and Sridhar Chimalakonda</i>	

An annotated repository for MATLAB code (S)	497
<i>Antonio Relvas, Nuno C. Marques, Miguel Monteiro and Glauro Carneiro</i>	

Session Fuzzy and Stochastic Approaches

BDFIS: Binary Decision Access Control Model Based On Fuzzy Inference Systems.	503
<i>Diogo Domingues Regateiro, Óscar Mortágua Pereira and Rui Aguiar</i>	

Fuzzy Bi-Objective Particle Swarm Optimization for Next Release Problem (S)	509
<i>Carlos Antonio Casanova Pietroboni, Giovanni Daian Rottoli, Esteban Schab, Luciano Bracco, Fernando Pereyra Rausch and Anabella De Battista</i>	

Language Independent POS-tagging Using Automatically Generated Markov Chains (S) . .	513
<i>Joaquim Assunção, Paulo Fernandes and Lucelene Lopes</i>	

Generating SQL Statements from Natural Language Queries: A Multitask Learning Approach (S)	518
<i>Chunqi Chen, Yunxiang Xiong, Beijun Shen and Yuting Chen</i>	

Session Software Systems I

Forward Engineering Completeness for Software by Using Requirements Validation Framework (S)	523
<i>Nayyar Iqbal, Jun Sang, Min Gao, Haibo Hu and Hong Xiang</i>	
Improving Mobile Device interaction for Parkinson's Disease Patients via PD-Helper.....	529
<i>Farzana Jabeen, Linmi Tao, Yirou Guo, Shiyu Zhang and Shanshan Mei</i>	
ACNET: Attention-based Convolution Network with Additional Discriminative Features for DCM Classification (S)	535
<i>Chao Luo, Wang Xin, Xiaojie Li, Yucheng Chen, Jiliu Zhou, Kunlin Cao, Qi Song, Xi Wu and Youbing Yin</i>	
Discovering Indicators for Classifying Wikipedia Articles in a Domain - A Case Study on Software Languages.....	541
<i>Marcel Heinz, Ralf Lämmel and Mathieu Acher</i>	

Session Mobile Software

Model View Controller in iOS mobile applications development	547
<i>Dragos Dobrean and Laura Diosan</i>	
An Empirical Study on Managing Energy and Accuracy Requirements of Location Based Android Applications (S)	553
<i>Marimuthu C, Sanjana Palisetti and Chandrasekaran K</i>	
LAD: A Layout Anomaly Detector for Android Applications	557
<i>Cheng-Zen Yang, Chih-Ju Lai, Peng Lu and Zhi-Jun You</i>	
Schedulability analysis for real-time mobile systems (S).....	563
<i>Cong Chen, Yangyang Chen, Jian-Min Jiang, Shi Zhang, Zhong Hong, Hongping Shu and Zeng Qiong</i>	

Session Requirements

Combining VSM and BTM to Improve Requirements Trace Links Generation	567
<i>Bangchao Wang, Rong Peng, Zhuo Wang and Yaxin Zhao</i>	
Themis: a tool for validating ontologies through requirements	573
<i>Alba Fernández-Izquierdo and Raúl García-Castro</i>	
Communication on Requirements Elicitation and Interaction Design through SPIDe (S) ..	579
<i>Jean Rosa, Beatriz Brito, Filipe Garrido, Pedro Valente, Nuno Nunes and Ecivaldo Matos</i>	

Impact of Agile Practices Adoption on Organizational Learning: a Survey in Brazil.....	583
<i>Florindo Silote Neto, Bruno Rafael de Oliveira Rodrigues, Renata de Souza França, Fabrício Ziviani and Fernando Silva Parreiras</i>	

Session Information Systems

Towards a customizable Student Information System integrating MDD and SPL (S).....	589
<i>Anderson Vale, Sergio Fernandes and Ana Patricia Magalhaes</i>	
Towards Detecting and Managing Information Anxiety in the ICT Industry	594
<i>Mark Micallef and Chris Porter</i>	
A Case Study of a Software Development Process Model for SIS-ASTROS.....	600
<i>Camila Hübner Brondani, Otávio da Cruz Mello and Lisandra Manzoni Fontoura</i>	

Session Ontologies

Experiences on applying SPL Engineering Techniques to Design a (Re) usable Ontology in the Energy Domain.....	606
<i>Javier Cuenca, Felix Larrinaga and Edward Curry</i>	
Identify MVC architectural pattern based on ontology.....	612
<i>Yin Qiang, Wang Lulu and Li Bixin</i>	
Grouping Semantically Related Change-Sets to Enhance Identification of Logical Coupling.....	618
<i>Neeraj Mathur, Sai Anirudh Karre and Raghu Reddy Y</i>	

Session Signal Processing

A Robust Visual Tracker Based on DCF Algorithm.....	624
<i>Menglei Jin, Weibin Liu and Weiwei Xing</i>	
A Novel Algorithm for Exemplar-based Image Inpainting (S)	630
<i>Yaru Cheng, Weibin Liu and Weiwei Xing</i>	
Trajectory Similarity Computation based on Interpolation and Integration (S).....	634
<i>Zengwei Zheng, Wenwang Chen, Yuanqi Chen and Dan Chen</i>	

Session Performance

Clustering algorithms performance analysis applied to patent database	640
<i>Cinthia M. Souza, Magali R. G. Meireles and Paulo E. M. Almeida</i>	
Automatic Calibration of Performance Indicators for Performance Analysis in Software Development (S)	646
<i>Mushtaq Raza and João Faria</i>	
Knowledge Engineering Research Topic Mining Based on Co-word Analysis.....	650
<i>Xiumin Liu and Zheng Liu</i>	
Finding Erroneous Components from Change Coupled Relations at Fix-inducing Changes	655
<i>Ali Zafar Sadiq, Ahmedul Kabir and Kazi Sakib</i>	

Session Software Quality

Artifact Quality Assessment Plans Generation from Tailored Processes	661
<i>Camila Hübner Brondani, Gelson Bertuol and Lisandra Manzoni Fontoura</i>	
Improve Language Modelling for Code Completion through Learning General Token Repetition of Source Code.....	667
<i>Yixiao Yang and Chen Xiang</i>	
Improve Language Modelling for Code Completion by Tree Language Model with Tree Encoding of Context (S)	675
<i>Yixiao Yang and Chen Xiang</i>	
Fast Exhaustive Search Algorithm for Discovering Relevant Association Rules	681
<i>Hend Amraoui, Faouzi Mhamdi and Mourad Elloumi</i>	

Session Software Process

Collecting Data from Continuous Practices: an Infrastructure to Support Team Development	687
<i>Ana Filipa Nogueira, Emilien Sergeant, Antoine Craske, José Carlos Ribeiro and Mário Zenha-Rela</i>	
Software Engineering Risks from Technical Debt in the Representation of Product/ion Knowledge	693
<i>Stefan Biffl, Lukas Kathrein, Arndt Lüder, Kristof Meixner, Marta Sabou, Laura Waltersdorfer and Dietmar Winkler</i>	
A Enhanced Feature Model for Software Product Line and Core Feature Extraction (S) ..	701
<i>Guanzhong Yang, Haoming Chang and Zeya Mou</i>	

Session Software Development I

SSLDoc: Automatically Diagnosing Incorrect SSL API Usages in C Programs	707
<i>Zuxing Gu, Jiecheng Wu, Chi Li, Min Zhou and Ming Gu</i>	
Multi-Location Program Repair Strategies Learned from Successful Experience (S)	713
<i>Shangwen Wang, Xiaoguang Mao, Nan Niu, Xin Yi and Guo Anbang</i>	
Logical Segmentation of Source Code.....	717
<i>Jacob Dormuth, Ben Gelman, Jessica Moore and David Slater</i>	
TL-GAN: Generative Adversarial Networks with Transfer Learning for Mode Collapse (S)	723
<i>Xianyu Wu, Shihao Feng, Xiaojie Li, Jing Yin, Jiancheng Lv and Canghong Shi</i>	

Session Software Systems II

Piecewise Aggregation for HMM fitting. A pre-fitting model for seamless integration with time series data.....	729
<i>Joaquim Assunção, Jean-Marc Vincent and Paulo Fernandes</i>	
How do Practitioners Manage Decision Knowledge during Continuous Software Engineering? (S).....	735
<i>Anja Kleebaum, Jan Ole Johanssen, Barbara Paech and Bernd Bruegge</i>	

Initial evaluation of the brain activity under different software development situations . . .	741
<i>Rustam Ikramov, Vladimir Ivanov, Sergey Masyagin, Ruslan Shakirov, Ilyas Sirazidtinov, Giancarlo Succi, Ananga Thapaliya, Alexander Tormasov and Oydinoy Zufarova</i>	
Finding conservative schema evolutions by analysing API changes	748
<i>Lynda Ait Oubelli, Yamine Ait-Ameur, Judicaël Bedouet, Benoît Chausserie-Laprée and Béatrice Larzul</i>	
<hr/> Session Software Systems III <hr/>	
Documenting and Exploiting Software Feature Knowledge through Tags	754
<i>Marcus Seiler and Barbara Paech</i>	
Research on Scheduling Area Partition Method Based on Multiple Algorithms	760
<i>Lefeng Li and Shanshan Wang</i>	
Anomaly Detection in the Registry of the Secondary Energy Distribution Network (S) . . .	766
<i>Carlos Fonsêca and Alexandre Maciel</i>	
Analyzing the impact of Technological KM and Participatory KM in FTA (S)	770
<i>Diego Cardoso Borda Castro, Carlos Eduardo Barbosa, Luis Felipe Coimbra Costa and Jano Souza</i>	
<hr/> Session Demo and Posters <hr/>	
Complex Networks Analysis for Software Architecture: a Case Study on Hibernate (P) . .	774
<i>Daniel Henrique Mourão Falci, Bruno Rafael de Oliveira Rodrigues, Orlando Abreu Gomes and Fernando Silva Parreiras</i>	
An Approach for Collecting Real Estate Development News (P)	776
<i>Vinícius Ferreira Salgado, Matheus de Oliveira Salim, Daniel Henrique Mourão Falci, Wladimir Cardoso Brandão and Fernando Silva Parreiras</i>	
GraphQL Servers generation from R2RML with morph-GraphQL (D)	777
<i>Ahmad Alobaid, David Chaves-Fraga, Freddy Priyatna and Oscar Corcho</i>	

Note: (S) indicates a short paper, (P) a poster and (D) a demo,

A DIK-based Question-Answering Architecture with Multi-sources Data for Medical Self-Service (KG)

Mengxing Huang^{1,2}, Menglong Li^{1,2}, YuZhang^{1,2}, Wenglong Feng^{1,2}

¹ State Key Laboratory of Marine Resource Utilization in South China Sea, Hainan University, Haikou, China

² College of Information Science & Technology, Hainan University, Haikou, China

Corresponding author: Yu Zhang (Email: yuzhang_nwpu@163.com)

Abstract—Medical data is amplified in terms of speed and capacity in a very fast way, which creates obstacles for users to quickly access valid information. We present a DIK-based Question-Answering Architecture for Medical Self-Service. In addition, we propose a model based on the attention mechanism to extract high-quality medical entity concepts from the Chinese Electronic Medical Records (EMR). Then we modeled the medical data based on the DIK architecture (Data graph, Information graph, and Knowledge graph), construct a Question-Answering model (DIK-QA) for medical self-service that meets the needs of users to quickly and accurately find the medical information they need in massive medical data. Finally, we have realized this approach and applied it to real-world systems. The experimental results on our medical dataset show that the DIK-QA can effectively handle 4W (who/what/why/how) questions, which can help users find the information they need accurately.

Keywords—Data graph, Information graph, and Knowledge graph (DIK); Question-Answering (QA); Electronic Medical Records (EMRs); attention mechanism

I. INTRODUCTION

Recently, knowledge graph have increasingly attracted attention in various fields. Medical Knowledge Graph (MKG) is also very popular in the medical field due to the unique expression of knowledge graph. Researchers explore with various approaches to construct the MKG. Reference [1] used deep learning method to extract 4 entities and 9 entity relationships from Chinese electronic medical records (EMR), then adopted the triple form to import data into the Neo4j and visualized the data into a knowledge graph. Reference [2] built a medical field table by using Bootstrapping and Conditional Random Fields (CRF) and solved the actual problem based on the obtained knowledge graph.

At present, research on knowledge graph is endless. Cowie et al. divided the knowledge graph into Data Graph (DG), Information Graph (IG) and Knowledge Graph (KG) according to the concept of Data, Information, Knowledge, and Wisdom (DIKW) [3].

However, there is a gap to fill for combining the MKG with the QA model. There are basically two major challenges. The first challenge is no effective framework for the union between QA and MKG for medical services. The second challenge the reliability of medical knowledge base. By addressing two major challenges, we solve and successfully construct a DIK-based QA for medical self-services. The complete structure of the DIK-QA is shown in Fig. 1. The contributions of our work are: 1) According to [4], we constructed the medical self-service DIK-QA based on the DIK. It can model massive amounts of data and quickly and accurately find the information users need and provide services to users in a friendlier manner, 2) We guarantee the data reliability of the medical knowledge base from two perspectives: clinical knowledge and health information. For clinical knowledge, we use the model BiLSTM-Attended-CRF model to obtain a higher quality medical entity concept. For health information we adopt the distributed crawlers to collect richer information, such as dietary advice and rehabilitation exercises.

II. FRAMEWORK

We divide the work into three parts: a) designing a DIK-QA framework (including three layers of knowledge graph), b) data acquisition, c) DIK-QA implementation. As shown in Fig. 1.

A. DIK-QA Architecture

According to the DIKW [3] idea, the medical knowledge graph can be divided into three categories: 1) Data graph, 2) Information graph, 3) Knowledge graph.

Data Graph Data graph can record the frequency of the data, including spatial frequency and structure frequency. We refer to the definition of data frequency in [4] to define the medical data frequency as a two-tuple.

$$DFreq = \langle f_{spatial}, f_{structure} \rangle \quad (1)$$

Where the spatial frequency, structure frequency of data are represented by $f_{spatial}$, and $f_{structure}$, respectively. The frequency of structure and spatial are the medical department of the disease and the treatment, respectively. As shown in Fig. 2.

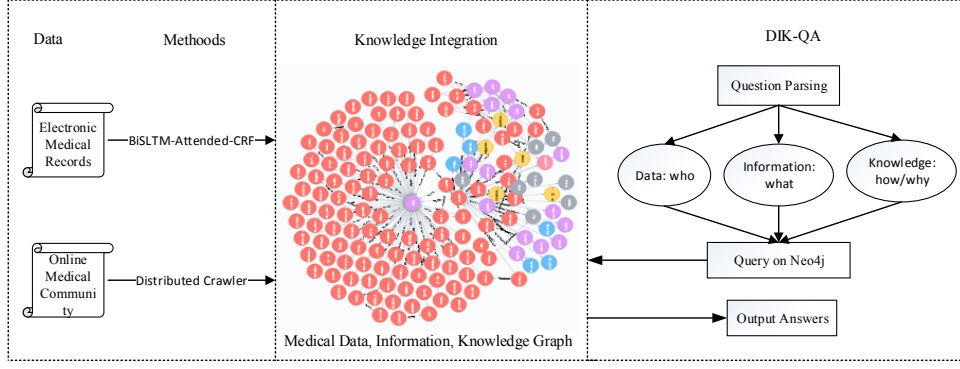


Figure 1. Overview of the DIK-QA Architecture for Medical Self-Service

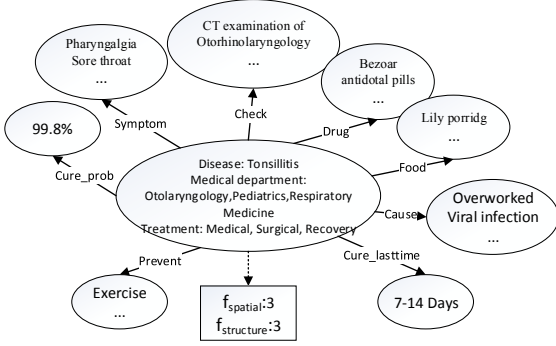


Figure 2. Statistics on $f_{spatial}$ and $f_{structure}$ of medical data.

Information Graph Information is extracted from the data for analysis and interpretation. We store the collected data as the medical knowledge base in dictionary format, as shown in Fig. 3. We define the medical knowledge base as the directed graph $G(V, E)$. The nodes and edges are denoted by V , E , respectively.[4]

$$Com_degree = \sqrt{deg^+ \times deg^-} \quad (2)$$

$$Impor = \alpha DFreq \times \beta Com_degree \quad (3)$$

The in-degree and out-degree of the node are represented by deg^+ , deg^- , respectively. In order to prevent the loss of the information, we adopt (3) to further measure the importance of the nodes. The weights of the $DFreq$ and the Com_degree is denoted by α , β .

Knowledge Graph Knowledge is the overall understanding and awareness gained from the accumulated information.

$$Cr(E_1, R, E_2) = \frac{\sum_{\pi \in Q} P(E_1 \rightarrow E_2) \theta(\pi)}{|Q|} \quad (4)$$

We use (4) to calculate the correctness of the relationship between E_1 and E_2 . Before we construct a medical knowledge graph, we need to design the knowledge graph: entity type nodes and entity relationship nodes, as shown in TABLE 1, 2.

Where all path between E_1 and E_2 are represented by Q , the weights of single path and path are represented by $\theta(\pi)$, π .

We comprehensively evaluate the importance of the nodes on the knowledge graph according to (5). N is the number of the relationship types. The weight of Rel_i is represented by λ_i .

$$Final_Impor = Impor \times \gamma \frac{\sum_{i=1}^n \lambda_i \times Rel_i}{n} \quad (5)$$

After completing the design of the DIK-based medical knowledge graph, we use the py2neo module in Python to import the dictionary type data into the Neo4j. As shown in Fig.1.

TABLE 1. The definition of the entity nodes

Entity nodes	Meaning
Department	Disease department
Disease	Disease name
Drug	Drugs for treating the disease
Cause	Cause of the disease

TABLE 2. The definition of the relationship nodes

Relation node	Meaning
do_eat	Recommended food for the disease
recommend_drug	Recommended drug for the disease
has_symptom	Disease corresponding symptoms
cure_way	Treatment for disease

B. Data Acquisition

In [1], we have extracted five entity types and nine entity relationship types by the BiLSTM-CRF model. However, we propose a novel model based on the Chinese EMR to improve the accuracy of the entity recognition, combining BiLSTM-CRF model with attention mechanism, as shown in Fig.4. Please refer to [1] for the BiLSTM-CRF model.

TABLE 3. The labeling rules of the Chinese EMR

Label Prefix	Label Suffix	Meaning
B	DISEASE,TREATMENT, CHECK, ,BODY, SIGNS	Head of the entity
I	DISEASE,TREATMENT, CHECK, ,BODY, SIGNS	Middle and tail of the entity
O	None	Other words

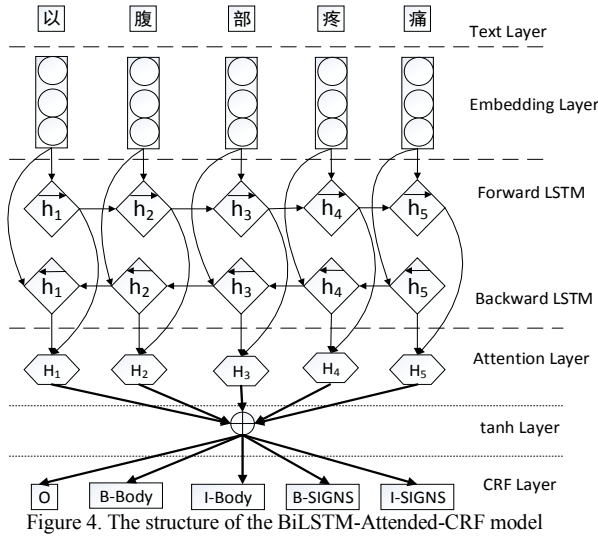


Figure 4. The structure of the BiLSTM-Attended-CRF model

BiLSTM-Attended-CRF Attention mechanism is a mechanism to simulate the attention of the human brain. The core idea is to draw on the attention of the human brain to things at a certain moment, it will focus on a certain key point, and ignore other key points [5]. We introduce the attention mechanism between the BiLSTM layer and the CRF layer to solve the challenge of extracting more precise semantic features for named entity recognition of Chinese EMR. Specifically, the attention mechanism is applied to the hidden layer of BiLSTM and then produces the newly hidden layer vectors. The implementation of the attention mechanism is as shown in (6)-(8).

$$e_{ij} = V \tanh(U_1 \vec{h}_i + U_2 \vec{h}_j + b_\alpha) \quad (6)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j=1}^T \exp(e_{ij})} \quad (7)$$

$$\vec{H}_i = \sum_{j=1}^T \alpha_{ij} \vec{h}_j \quad (8)$$

Where e_{ij} represents energy value of the j th word to the i th word, that is to say, the relationship between the two words. We use α_{ij} represents the attention weight of the j th word to the i th word, reflecting the influence of each vocabulary in the input text X . V , U_1 , U_2 , b_α is the parameter trained with the model. The new feature representation after the attention mechanism in Forward-LSTM Layer and Backward-LSTM is represented by \vec{H}_i , respectively. We combine two new feature representations to form the final feature representation $H_i = [\vec{H}_f, \vec{H}_b]$. After the attention layer, it will pass the activation function tanh layer, then pass the mapping results of tanh layer to the CRF Layer. Finally, the model can output the predicted label sequence for input text, as shown in Fig. 4.

Disease-Oriented We constructed a medical entity dictionary for recognition of the BiLSTM-Attended-CRF model. We take the knowledge fusion based on the medical concept in the dictionary to eliminate some ambiguous entities. The Disease-Oriented Knowledge base has the advantages: the disease can integrate various information

such as symptoms, checks, causes, foods and drugs et al. into one line, as shown in Fig. 2.

C. DIK-QA Implementation

The DIK-QA model is consist of three parts: Analysis question, Question division, Query knowledge. As show in Fig.1.

Analysis question The DIK-QA can analysis the question in plain language way by the BiLSTM-Attended-CRF model. In this part, we can get the medical entity in the question, then we submit the identified results to the next part-Question division.

Question division After the analysis question, we can get the category of question. By classifying the user's questions, we can accurately get the intent of user. According to the frequency statistics on the types of questions and vocabulary that appear in the Q&A module on the medical website, we summarize two main types of questions: 1) Entity relationship template. It can solve the question of entity relationship query, including disease and symptoms, disease and drug, food, treatment and so on. For example, "what are the symptoms of hypertension?", "Recently dizzy, what disease might I get?", etc. 2) Entity attribute template. It can solve the question of entity attribute query, including disease prevention, cause and so on. For example, "why do I get the hypertension?", "Precautionary measures for the hypertension?".

Query knowledge We can take the query on the Neo4j based above part. We take the Aho-Corasick (AC) algorithm [6] as the core method to realize the Query knowledge. AC automaton is the most classic multi-pattern matching algorithm. It uses a plurality of pattern strings to construct a finite state pattern matching automaton. The DIK-QA performs knowledge reasoning on the corresponding knowledge graph (DIK) according to different question types, so that the answer can be quickly found. We can convert the question to cypher query statement and search the corresponding answer, and then return it. Cypher is a graph query language designed for operating Neo4j that efficiently queries and updates knowledge graphs. It has more powerful than SQL in relational capabilities. For example, Question: what are the symptoms of tonsillitis? Cypher statement: Match (m: tonsillitis) -[r: has_symptom] → (n: Symptom) where m.name = '{0}' return m.name, r.name, n.name. Where r, m, n are the variables of the disease name, relation: has_symptom, and symptom node, respectively. Node definition refer to TABLE 1, 2.

III. EXPERIMENTS AND RESULTS

A. Experiments on BiLSTM-Attended-CRF

To evaluate the performance of the BiLSTM-Attended-CRF model, we compared several basic model. The EMR data

from the local hospital. Other medical data comes from two medical websites¹. The evaluation indicator adopts accuracy, recall rate, and F value. The comparison algorithm uses common models such as CRF, BiLSTM, and BiLSTM-CRF. We tested 100, 128, 200, 256 and 300 on word vector dimension. The BiLSTM-Attended-CRF model has a good performance when word vector dimension is 128. Learning rate is 0.001, the LSTM layer is 4. Dropout is 0.4. The experimental results are shown in Table 4.

TABLE 4. Experimental results of each model for entity recognition

Model	P (%)	R (%)	F (%)
CRF	60.45	58.72	60.13
BiLSTM	61.42	58.23	57.68
BiLSTM-CRF	83.53	78.38	80.35
BiLSTM-Attended-CRF	89.76	85.51	88.90

The experimental results show that the BiLSTM-Attended-CRF model has better performance than state-of-the-art baselines in entity recognition. F-value and accuracy increased by 6%. In our model, all three evaluation marks have been improved.

B. Experiments on the DIK-QA

To evaluate the effectiveness of the DIK-QA, we selected five common diseases as small sample data, including cold, pediatric cold, hypertension, diabetes, cervical spondylosis. Observed from Fig.5, we can know that the DIK-QA can solve effectively some common question.

<p>DIK-QA work situation</p> <p>...</p> <p>User: 如何治疗感冒? (How to cure a cold?)</p> <p>Doctor: 感冒可尝试如下治疗: 中医治疗, 支持性治疗, 对症治疗。(You can try the treatments for cold: Chinese medicine treatment, supportive treatment, symptomatic treatment.)</p> <p>User: 感冒的症状都有什么? (What are the symptoms of a cold?)</p> <p>Doctor: 感冒的症状有: 鼻塞, 咽痛, 发热, 头痛等。(The symptoms of a cold are nasal congestion, sore throat, heat, and headache etc.)</p> <p>User: 感冒忌吃什么食物? (What food can't you eat when you have a cold?)</p> <p>Doctor: 感冒忌吃: 猪油, 咸鱼, 白扁豆, 油条等。(Avoid eating: lard, salted fish, and fritters, etc)</p> <p>User: 感冒应该吃什么食物? (What food should I eat for a cold?)</p> <p>Doctor: 感冒应该吃: 鸡蛋, 芝麻, 南瓜子仁, 鹌鹑蛋。(Recommend eating: egg, pumpkin seed kernel, sesame, quail eggs)</p> <p>User: 为什么会感冒? (Why do people catch a cold?)</p> <p>Doctor: 感冒的可能成因: 病毒感染 (70%-80%), 有受凉、淋雨、过度疲劳等诱发因素, 导致抵抗力下降。(The possible causes of a cold: viral infection(70%-80%), or factors such as cold, rain, excessive fatigue, etc., which can reduce the resistance...)</p> <p>...</p>
--

Figure 5. The DIK-QA's work situation

IV. RELATED WORK

In 2006, Rowley [7] revisits the data-information-knowledge-wisdom (DIKW) hierarchy and analyses the statement about the nature of data, information, knowledge, and wisdom. Through this process we get a consensus on definitions and transformation about DIKW. It makes the theoretical development of the further development of the DIKW. Duan et al. [8] proposes to clarify the expression of knowledge graph as a whole for lacking a unified definition and standard expression form of knowledge graph. They clarify the architecture of knowledge graph from data, information, knowledge and wisdom aspects respectively. They also propose to specify DIKW-based knowledge graph. Moreover, Song et al. [9] proposed a processing optimization mechanism of typed resources in a wireless-network-based

three-tier architecture consisting of DIK mechanism. Simulation results show that the proposed approach improve the ratio of performance over user investment.

V. CONCLUSIONS

The main task of this paper is: 1) we propose a highly accurate medical entity recognition model--BiLSTM-Attended-CRF to extract the high-quality medical concepts, 2) we construct the DIK-QA for medical self-service. The DIK-QA is mainly focused on some common diseases and questions such as colds, headaches, fever, ligament strains and some special diseases: pneumonia, etc.

However, there are still some shortcomings in this paper. We can combine DIK-QA with the deep learning method to automatically analyze the problem. Then, we can construct a smarter medical self-service. We can also try to construct a wider application of the DIKW architecture.

ACKNOWLEDGMENT

This work was supported by the Key R&D Project of Hainan province (Grant #: ZDYF2019020), National Natural Science Foundation of China(Grant #: 61662019 and 61862020), Major Science and Technology Project of Hainan province (Grant #: ZDKJ2016015), Higher Education Reform Key Project of Hainan province (Grant #: Hnjg2017ZD-1).

REFERENCES

- [1] M.X. Huang, M.L. Li, H.R. Han, "Research on entity recognition and knowledge graph construction based electronic medical records," in press.
- [2] Y.B. Zhang, Research on the construction of medical knowledge graph and its application. Ph.D. Thesis, Harbin Institute of Technology. Harbin, China: Haibin Institue of Technology, 2018.
- [3] J. Cowie, W. Lehnert, Information extraction. Berlin, Hei-delberg: Springer, 2004.
- [4] L.X. Shao, Y.C. Duan, Z.B. Zhou, et al., "Design of recommendation services based on data, information and knowledge graph architecture," Journal of Frontiers of Computer Science and Technology, Biejin, China, vol. 13, 2019, pp.214 - 225.
- [5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez et al., "Attention is all you need," *Advances in Neural Information Processing Systems* 2017, pp.5998-6008.
- [6] A.V. Aho, M.J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, 1975, pp.333-340.
- [7] Rowley, Jennifer (2007). "The wisdom hierarchy: representations of the DIKW hierarchy". *Journal of Information and Communication Science*. 33 (2): 163–180.
- [8] Y.C. Duan, L.X. Shao, G.Z. Hu, "Specifying Knowledge Graph with Data Graph, Information Graph, Knowledge Graph, and Wisdom Graph," *IJSI* vol.6, 2018, pp.10-25.
- [9] Z.Y. Song, Y.C. Duan, S.X. Wan, X.B. Sun, Q. Zou, H.H. Gao, D.H. Zhu, "Processing Optimization of Typed Resources with Synchronized Storage and Computation Adaptation in Fog Computing," *Wireless Communications and Mobile Computing*, 2018, pp.1-13.
- [10] L.X. Shao, Y.C. Duan, L.Z. Cui, Q. Zou, X.B. Sun, "A Pay as You Use Resource Security Provision Approach Based on Data Graph, Information Graph and Knowledge Graph," *IDEAL*, 2017, pp.444-451.
- [11] Y.C. Duan, G.H. Fu, N.J. Zhou, "Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends," *IEEE 8th International Conference on Cloud Computing (CLOUD)*. IEEE, 2015.

¹ <http://www.xywy.com>

Constructing a Knowledge Base of Coding Conventions from Online Resources

Junming Cao, Tianjiao Du, Beijun Shen*, Wei Li, Qinyue Wu, Yuting Chen

School of Electronic Information and Electrical Engineering

Shanghai Jiao Tong University, Shanghai, China

{junmingcao, tjsoulshe, bjshen, li_wei, wuqinyue, chenyt}@sjtu.edu.cn

Abstract—Coding conventions are a set of coding guidelines used by software developers to improve the readability of source code, increase software maintainability, and promote the reuse of coding patterns. In this paper, we introduce *CCBase*, a knowledge base of coding conventions, that was constructed from online resources. Specifically, *CCBase* was constructed as follows. We designed the ontology of the coding convention domain, crawled data related to coding conventions from a variety of online resources, and then extracted entities and relations using an NLP-enabled rule matching method. To uncover the latent relations, we further proposed a similarity metric to reveal the similar-to and relate-to relations, and developed a RCE algorithm to establish a unified type hierarchy of coding conventions. The resulting knowledge base contains 3139 coding conventions for Java and C++, with 3761 entities and 767 relations. Furthermore, we have extended the usability of *CCBase* by developing a question answering system on the base. We have conducted experiments to evaluate *CCBase*. The experimental results show that *CCBase* has a wide coverage on entities and relations in coding conventions domain, and the QA system achieves an F1 score of 84.5% on 214 questions raised in StackOverflow.

Keywords - *Knowledge Base; Coding Convention; Type Hierarchy; Question Answering*

I. INTRODUCTION

Coding conventions are a set of guidelines for a particular programming language that recommend programming styles, practices, and methods for each aspect of a program written in that language. During increasingly large and complex software development, programmers are strongly encouraged to follow these guidelines to help improve the readability, reliability, and maintainability of their source code [1]. These coding conventions can also assist code related software engineering activities, like auto-detection of code bad smells [2] and code analysis [3].

However, programmers now encounter the following problems when applying coding conventions. One is that coding conventions specified in single document are incomplete because it could hardly cover a • ll coding details, and also the relations between coding conventions could not be expressed explicitly. The second problem is that coding conventions are inconvenient to access. Programmers need to know relevant keywords to search using a search engine like Google or search

in documents, which is especially difficult for some novice programmers who lack professional knowledge.

In order to solve the above problems, we construct a coding conventions knowledge base, *CCBase*. *CCBase* is a domain-specific knowledge base, which is constructed from online resources using a top-down approach. Specifically, we first design the ontology of coding conventions domain. Then we collect data related to coding conventions from various online resources and extract entities and relations with an NLP-enabled rule matching method. The main challenge is to discover latent relations between coding conventions, including similar-to, relate-to, and especially subsumption relations, from these heterogeneous textual documents. So we designed a similarity metric to discover similar-to and relate-to relations, and propose the RCE (Relation based Cluster Expansion) algorithm to establish a unified type hierarchy of coding conventions and assign types of each coding convention. Finally, we develop a question answering system over *CCBase* to answer natural language questions automatically. *CCBase*, its SPARQL interface, and QA system can be accessed in our online platform¹.

Our main contributions are summarized as follows:

1) To our best knowledge, *CCBase* is the first knowledge base of coding conventions. It contains 3139 coding conventions of Java and C++, 3761 entities and 767 relations.

2) We propose the RCE algorithm to establish a unified type hierarchy of coding conventions. Structures of online resources entail original type hierarchies for coding conventions. However, some coding convention resources lack a hierarchy. The hierarchy extracted from one document is usually unilateral, and also different from another extracted hierarchy. Besides, every coding convention only has one type value with the original type hierarchy, which is also not comprehensive. Therefore, a novel unsupervised algorithm (RCE) is designed to build a unified type hierarchy according to the similar-to relations and assign new type values to coding conventions.

3) We develop a coding convention question answering system over *CCBase*, *CCQA*. The main algorithm of *CCQA* is subgraph matching, and we make two significant improvements to this algorithm. First, the entity linking method is changed to identify the entities regarding coding conventions in the question. Second, we collect common question templates and recognize

DOI reference number: 10.18293/SEKE2019-123

* Corresponding author

¹ <http://202.120.40.28:4463/>

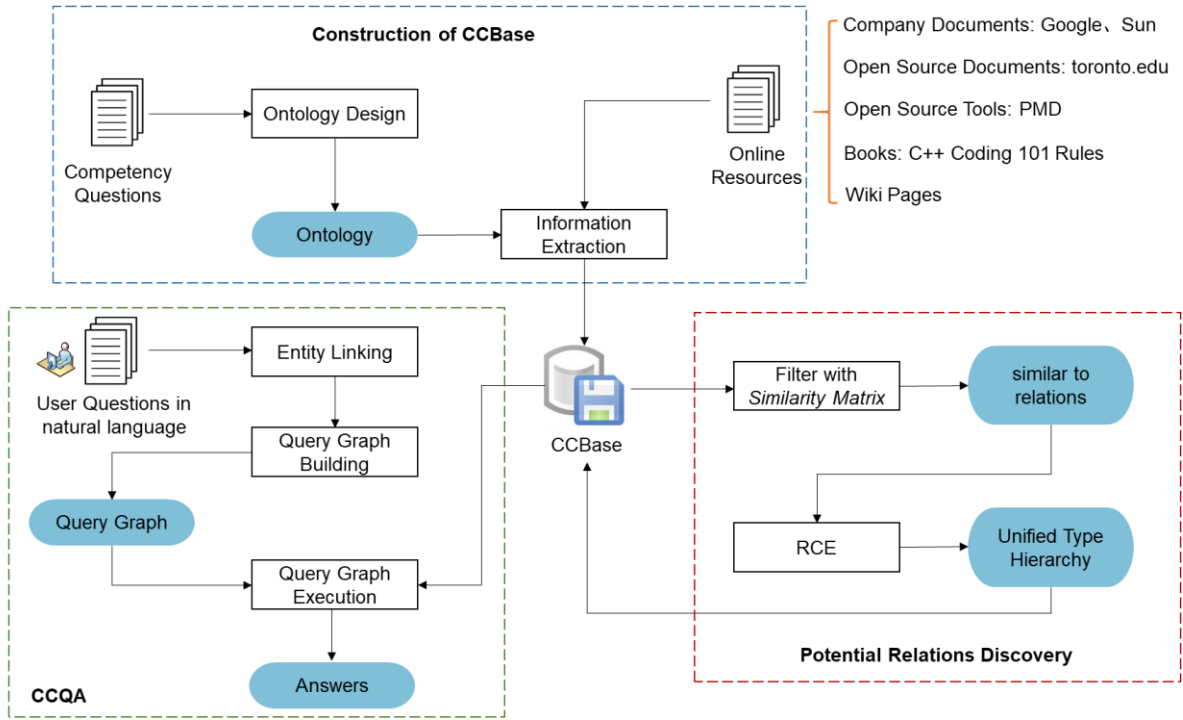


Figure 1. Overview of Our Approach

these templates from user questions, which improves the accuracy of subgraph building

4) A set of comprehensive experiments has been carried out to evaluate CCBASE. The results show, CCBASE is larger and more hierarchical than existing knowledge bases regarding code conventions; and our QA system achieves an F1 score of 84.5% on 214 questions raised in StackOverflow.

II. RELATED WORK

A. Construction of Knowledge Base

There are two ways to construct a knowledge base: top-down and bottom-up. Top-down means pre-defining the ontology of a knowledge base, and then importing entities and relations according to the ontology into the knowledge base. Knowledge bases of specific domains mostly adopt this way [5]. Bottom-up means directly obtaining entities and relations by syntactic analysis without ontology, which is common for general knowledge bases [6]. These two ways include similar steps such as information extraction and knowledge fusion.

In the software engineering domain, a few researchers have tried to build a domain knowledge base, like SEBase [5] and APIBase [7]. However, to our best knowledge, there is no published work on coding convention knowledge base.

B. Type Hierarchy Building

Types are common in knowledge bases to organize entities, and type hierarchy is their key knowledge or meta-knowledge. [8] proposed an entity-driven approach to construct type hierarchy of knowledge base systems without hierarchy structures. The type hierarchy construction problem is similar to the community detection problem. Semi-supervised algorithms, like LPA [9], are widely used in community detection, and [10]

proposed SLPA to deal with overlapping communities. However, these methods aren't suitable for our work, because our relations in CCBASE are too sparse to propagate labels from a few seed entities, and structures of documents are very helpful to build the type hierarchy. Thus in this paper, we propose a novel unsupervised algorithm (RCE) by fully utilizing structures of documents.

C. Question Answering over Knowledge Base

Some research effort has been conducted to KBQA (Question Answering over Knowledge Base) systems [12][13], which led to major advances. So far there exist two mainstreams of KBQA methods. One mainstream is semantic parsing. The main idea of this kind of solution is to translate the questions into logical forms such as query graph, then generate executable queries [12]. Information retrieval is another mainstream, which selects candidate answers directly and then ranks these answers by various approaches, such as deep learning [13].

Our work belongs to the first one. Since natural language is complex and ambiguous, semantic parsing usually requires multiple steps, like part-of-speech tagging and entity linking.

III. CONSTRUCTION OF CCBASE

We construct CCBASE in a top-down way for the following reasons: 1) Bottom-up is difficult to meet the quality requirements of domain-specific knowledge base. 2) The complexity of entities and relations in the coding conventions domain is tractable enough to be designed in advance. 3) Entities and relations could not be automatically obtained by syntactic analysis, so ontology is necessary to guide the extraction of entities and relations.

The overall approach is shown in Fig. 1. We first design the ontology of CCBBase, then extract the information from the semi-structured and unstructured data, and finally discover the latent relations between coding conventions.

A. Ontology Design

We collect massive coding conventions from various online resources, including coding conventions published online by companies, standards organizations, research groups and experts, coding conventions in open source tools, books, wiki pages, etc. The ontology is initialized from the investigation of these data. We use Protégé², an open source software developed by Stanford, to design the ontology.

Then we use the competency question-driven method to perform ontology improvement [14]. We select 30 competency questions from 214 coding convention related questions from StackOverflow, and improve the ontology until these questions could be answered with the ontology. The final ontology has 11 key concepts and 14 kinds of relationships, as shown in Fig. 2.

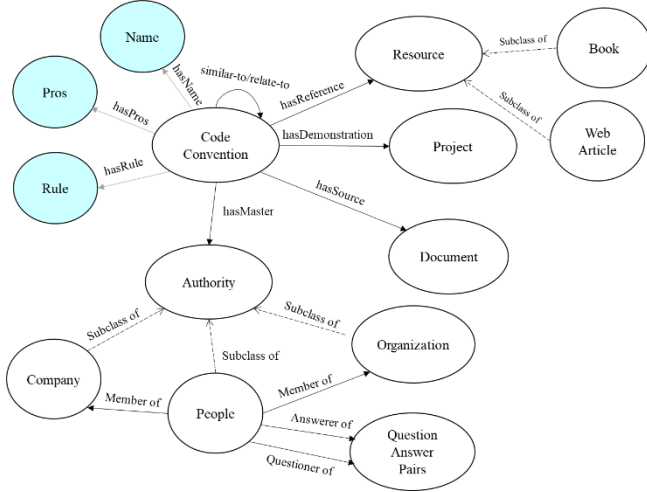


Figure 2. Ontology of CCBBase

B. Information Extraction

Guided by the ontology, we extract instance data from collected textual materials and store them in CCBBase.

The syntactic analysis approach [15] is widely used to extract <subject, predicate, object> triples from sentences, but it isn't applicable for constructing CCBBase. It is because entities and property values of coding conventions could not be directly collected from sentences, and also predicates in triples parsed by syntactic analysis approach could not be used as relations in CCBBase. Therefore, we propose a semi-automated method to import entities and relations into CCBBase, which consists of four steps:

- 1) Parse file structures of documents.
- 2) Define a set of rules based on keywords like "example", "benefits" and "author" to match candidates of entities, relations, and properties of entities.
- 3) Extract candidate entities and relations by rule matching.

² <https://protege.stanford.edu/>

- 4) After quality checking by experts, the final results are imported into CCBBase.

The semi-automated method is more accurate than the syntactic analysis method, while it does not cost as much as fully human collection method. Fig. 3 shows some entities and relations gained from information extraction, except for similar-to and relate-to relations, which would be discovered further in section C.

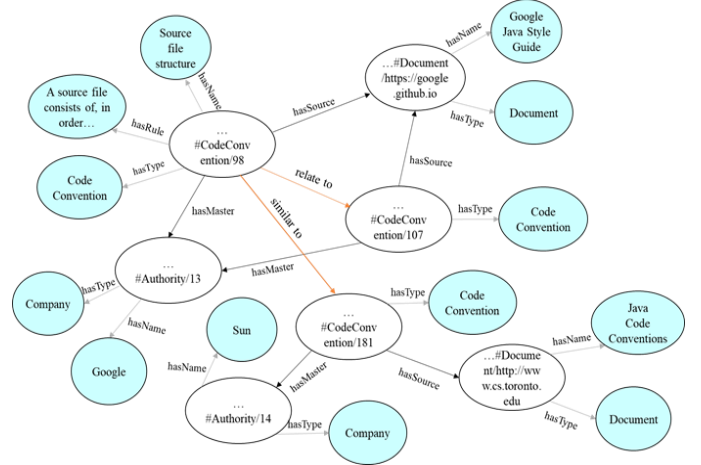


Figure 3. One Fragment of Instances in CCBBase

C. Relation Discovery

It is necessary to further discover the latent relations between entities. According to the ontology structure of CCBBase, the relations between entities include the relations between different coding conventions, and relations between coding conventions and other types of entities. The latter, like hasSource and hasMaster in Fig. 2, could be obtained through information extraction. Thus we focus on discovering latent relations between coding conventions.

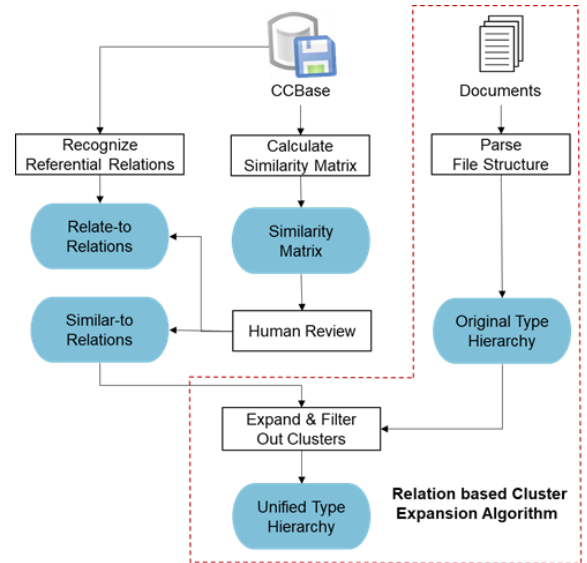


Figure 4. Relation Discovery

We propose a semantic similarity measuring approach to discover these following relations, as shown in Fig. 4.

1) **similar-to**. [16] lists a set of widely accepted metrics to measure the similarity between entities. Considering most properties of entities in CCBase are long texts, we adopt WHIRL as the similarity metric, which is based on TF-IDF. Then we set up a threshold by experiments to obtain the *Similarity Matrix*, which contains entity pairs with high WHIRL metric values. Finally, experts decide on whether entity pairs in *Similarity Matrix* have similar-to relations.

2) **relate-to**. There are two types of relate-to relations in CCBase. One is referential relations between coding conventions from the same document. The description of a coding convention may refer to other coding conventions in the same document. For example, the coding conventions named "Package Statement" in the Google Java Style Guide is described as "The package statement is not line-wrapped. The column limit (Section 4.4, Column limit: 100) does not apply to package statements." It refers to the coding convention named "Column limit: 100". For this type of relations, we could find them through information extraction. Another kind of relate-to relations come from entity pairs in the *Similarity Matrix* that do not have similar-to relations.

3) **subsumption**. There is an original type hierarchy of code conventions in each document. For example, coding convention named "Naming Convention" includes "Function Naming Convention", "Variable Naming Convention", etc. However, original type hierarchies have three shortcomings as described in the introduction section. Thus, we propose the RCE algorithm to establish a unified type hierarchy for coding conventions from all documents.

Algorithm: RCE

Input: Given entity set E , document set D , original type hierarchy set S . S_{ij} is the j_{th} primary type of type hierarchy S_i from D_i and S_{ijk} is the k_{th} secondary type belonging to S_{ij} .

Procedure:

1: Expand candidate entity clusters with the same type C according to relations.

```

for  $i, j$  in range(0, length( $D$ )), range(0, length( $S_i$ )):
     $C_{ij}.entity \leftarrow \emptyset$ 
     $C_{ij}.layer \leftarrow primary$ 
    for  $k$  in range(0, length( $S_{ij}$ )):
         $C_{ijk} = \text{Entities of } S_{ijk}$ 
        for  $e$  in Entities of  $S_{ijk}$ :
             $C_{ijk}.entity \leftarrow C_{ijk}.entity \cup similarEntities(e)$ 
             $C_{ijk}.layer \leftarrow secondary$ 
             $C_{ij}.entity \leftarrow C_{ij}.entity \cup C_{ijk}.entity$ 

```

2: Filter out replicated clusters and clusters with too few entities. Name every cluster to get type hierarchy R .

```

for  $c_1, c_2$  in  $C$ :
    if similarity( $c_1, c_2$ ) >  $\theta$ :
         $C.remove(c_2)$ 
        similarity( $c_1, c_2$ ) =  $|c_1.entity \cap c_2.entity| / |c_1.entity \cup c_2.entity|$ 
for  $c$  in  $C$ :
    if  $|c| < \delta$ :
         $C.remove(c)$ 
    else:
         $t.entity \leftarrow c.entity$ 

```

```

 $t.layer \leftarrow c.layer$ 
 $t.name \leftarrow \text{select one name from original types of } t.entity$ 
or make a new name
 $T.push(t)$ 

```

3: Generate type lists for entities.

```

for  $t$  in  $T$ :
    for  $e$  in  $t$ :
        if  $t.layer$  is primary:
             $e.primary\_type\_list.push(t.name)$ 
        else:
             $e.secondary\_type\_list.push(t.name)$ 

```

Output: Unified Type hierarchy T , entities set E' with primary and secondary type lists.

The unified type hierarchy T holds two layers: the primary layer, and the secondary layer. As we expand clusters with similar-to relations in Step 1, some entities would belong to multiple clusters and finally multiple types, like entities in Freebase. Thus, we use lists to store primary types and secondary types in Step 3. Although there are no direct relations between entities that share the same types, we could group these entities easily by type. This is the reason that we take it as a kind of relation. As a result, a unified type hierarchy for coding conventions is built with 16 primary types and 53 secondary types, as shown in Fig. 5.

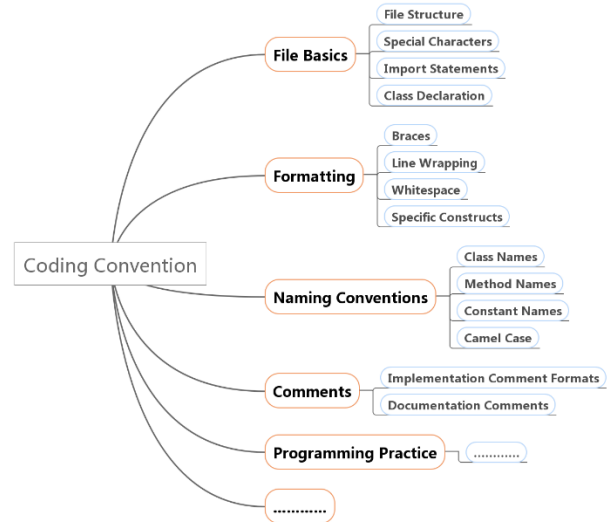


Figure 5. Part of the Unified Type Hierarchy

When applied in CCBase, RCE has the following advantages over LPA and LPA-based algorithms, like SLPA:

- As LPA-based algorithms are semi-supervised, they need many labeled seeds or dense relations between entities to propagate labels, but relations in CCBase are too sparse. Since RCE is unsupervised, it does not suffer from this problem.
- RCE fully utilizes original type hierarchies of documents, while only one layer of original type hierarchies could be used as labels in LPA-based algorithms.

IV. QUESTION ANSWERING OVER CCBASE

To demonstrate the value of CCBASE, we develop a question answering system over it, called CCQA. It can assist programmers to retrieval information about coding conventions in a more natural manner.

Inspired by Hu et al.’s work [4], we propose the LE (long entity) Node-First framework to answer coding convention questions by subgraph matching. As Fig. 1 shows, we first extract semantic relations based on the dependency tree of question sentences to build a semantic query graph Qu . A semantic relation is a triple $\langle rel; arg1; arg2 \rangle$, where rel is a relation phrase, and $arg1$ and $arg2$ are its associated node phrases. After that, a SPARQL query statement is generated from Qu and then executed to get final answers.

LE (long entity) Node-First framework improves Hu et al.’s work from the following two points.

First, since entities about coding conventions are usually complete sentences instead of words or phrases, we use Jena Full Text Search and combine rule-based method for entity linking. We merge words within specific property of entities into one node to obtain clearer sentence structures, and thus the further generated dependency tree can achieve higher accuracy.

Second, when building a query graph Qu , the algorithm of [4] also extracts wh-words (what, how, why etc.) as nodes. However, if a question only contains one entity and does not contain any wh-word or relation, the query graph Qu will only be formed as one node and the query will be failed. Thus it could not answer Yes/No questions and declarative sentence. To improve the ability of CCQA, we collect some common question templates, such as questions begin with “*Is there any*”. These templates will also be recognized as nodes from questions.

So far CCQA has been developed as a plugin in IntelliJ IDEA, which can be downloaded from our Github project³.

V. EVALUATION

Several experiments have been conducted to evaluate CCBASE and CCQA.

A. Performance of Information Extraction

We construct CCBASE in a top-down way, extracting entities and relations from unstructured documents guided by ontology. To evaluate the effectiveness of this method, we compare it with two bottom-up extraction methods: the popular open information extraction tool – *open IE* [20] and a domain-specific extraction method – HDSKG [15]. Three popular metrics are selected: precision, recall and F1 score.

We collect 8 documents about coding conventions as the dataset of this experiment, and then we ask three experts to label the data manually. Fig. 6 shows the results of the comparison. Open IE and HDSKG both extracts the dependencies from sentences to generate relation triples. However, the entities and relations in coding convention domain are too complex to be directly extracted from one single sentence. The top-down extraction method outperforms HDSKG by 44.2% in F1 score.

Furthermore, we also conduct an experiment to compare the algorithms of relation discovery. We use LPA, SLPA, and RCE to build different versions of knowledge bases. The results are shown in Fig. 7. We can find that SLPA and RCE perform much better than LPA, because types generated by LPA do not overlap, which is unreasonable for coding conventions. Benefiting from the original type hierarchies of documents, RCE outperforms SLPA by 4.3% in F1 score.

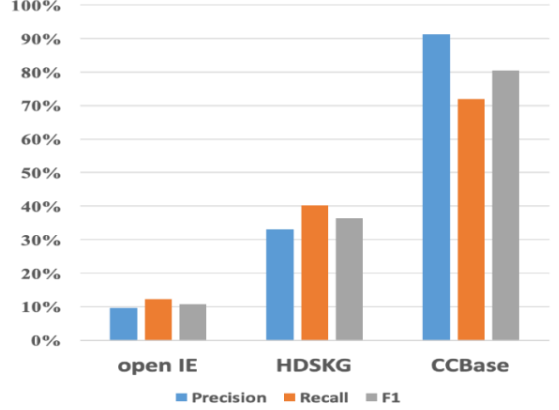


Figure 6. Evaluation of Information Extraction Methods

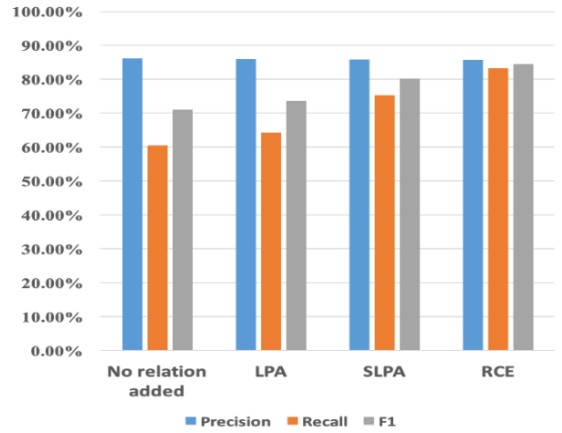


Figure 7. Evaluation of Relation Discover Methods

B. Comparison with other Knowledge Bases

As there are no public knowledge bases in the field of coding convention, we compare CCBASE with related subsets of a software engineering knowledge bases such as SEBase [5] and software.zhishi.schema [19]. We also compare it with YAGO [18], a general knowledge base.

TABLE I: Comparison with Other Knowledge Bases

	CCBASE	SEBase	zhishi	YAGO
Concept	3761	128	38	31
Subsumption	181	57	50	0
Relate-to	524	12	0	0
Similar-to	62	0	0	0

³ <https://github.com/14dtj/code-convention-robot>

Table I shows the number of entities and relations of each dataset. We could discover that our knowledge base is larger than other existing datasets as for the entity number related to coding conventions. Besides, the relations between entities are richer, especially as for subsumption and related-to relations.

C. Performance of Question Answering

We crawled 214 code convention questions from StackOverflow as experimental datasets. The performance of a QA system is measured by the ratio of questions that are answered correctly.

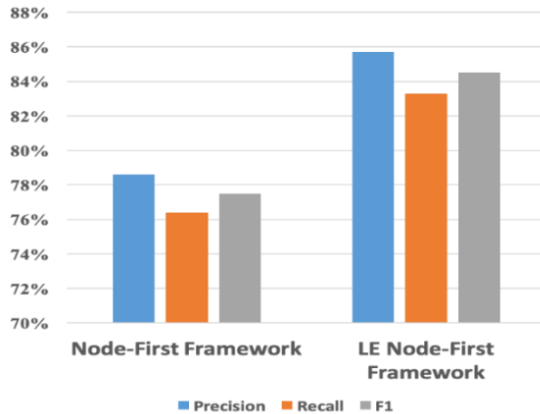


Figure 8. Evaluation of Question Answering Methods

We compare our approach (LE Node-First Framework) with Node-First Framework by Hu et al. [4]. Fig. 8 reveals the results on 214 questions. Node-First Framework adopts CrossWikis dictionary [17] to map entities in user questions, which is not suitable for long entity linking. Besides, it could not handle Yes/No questions and declarative sentences. It is shown that our LE Node-First Framework achieves 84.5% in F1 score, while the F1 score of original Node-First Framework is only 77.5%.

VI. CONCLUSION

In this paper, we designed and constructed CCBASE, the first coding convention knowledge base, from online resources. And for programmer's convenient access, a question answering system over CCBASE was further developed. Experiments show that CCBASE contains much more entities and relations about coding conventions than previous knowledge bases, and our QA system achieves an F1 score of 84.5% on 214 questions raised in StackOverflow.

As for future work, we will try to extract more entities and relations about coding conventions from Github and other Web sites to enrich CCBASE. Moreover, it would be interesting to explore more potential applications based on this CCBASE such as code bad smell detection.

VII. ACKNOWLEDGEMENT

This research was sponsored by the National Key Research and Development Program of China (Project No. 2018YFB1003903), National Nature Science Foundation of China (Grant No. 61472242 and 61572312), and Shanghai Municipal Commission of Economy and Informatization (No. 201701052).

REFERENCES

- [1] Bahman Arasteh, Jalal Najafi, "Programming guidelines for improving software resiliency against soft-errors without performance overhead", *Computing* 100(9): 971-1003 (2018)
- [2] CHEN, H., CHEN, W., and Lee, C. C. (2018). "An Automated Assessment System for Analysis of Coding Convention Violations in Java Programming Assignments", *Journal of Information Science and Engineering*, 34(5), 1203-1221.
- [3] Tourwe, Tom, and Kim Mens, "Mining aspectual views using formal concept analysis.", *Source Code Analysis and Manipulation, Fourth IEEE International Workshop on*, 2004, pp. 97-106.
- [4] Hu S, Zou L, Yu J X, et al, "Answering natural language questions by subgraph matching over knowledge graphs" in *IEEE Transactions on Knowledge and Data Engineering*, 2018, 30(5): 824-837.
- [5] Kai Chen, Xiang Dong, Jiangang Zhu, Beijun Shen, "Building a Domain Knowledge Base from Wikipedia: a Semi-supervised Approach", *SEKE*, 2016, pp. 191-196
- [6] Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge", in *SIGMOD '08 Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247-1250.
- [7] Hongwei Li, Sirui Li, Jiamou Sun, Zhenchang Xing, Xin Peng, Mingwei Liu, Xuejiao Zhao, "Improving API Caveats Accessibility by Mining API Caveats KnowledgeGraph", *ICSME 2018*, pp. 183-193.
- [8] Jiang, Jyun-Yu, Pu-Jen Cheng and Chin-Yew Lin, "Entity-driven Type Hierarchy Construction for Freebase.", *WWW, 2015*, pp. 47-48.
- [9] J. Xie and B. K. Szymanski, "Community detection using a neighborhood strength driven label propagation algorithm," in *Network Science Workshop (NSW), 2011 IEEE*, pp. 188-195.
- [10] J. Xie, B. K. Szymanski and X. Liu, "SLPA: Uncovering Overlapping Communities in Social Networks via a Speaker-Listener Interaction Dynamic Process," in *IEEE 11th International Conference on Data Mining Workshops, Vancouver, 2011*, pp. 344-349.
- [11] W. Yih, M. Chang, X. He, and J. Gao, "Semantic parsing via staged query graph generation: Question answering with knowledge base," in *Proc. 53rd Annu. Meet. Assoc. Comput. Linguistics 7th Int. Joint Conf. Natural Language Process. Asian Fed. Natural Language Process.*, 2015, pp. 1321-1331.
- [12] C. Unger, L. Böhmann, J. Lehmann, A.-C. N. Ngomo, D. Gerber, and P. Cimiano, "Template-based question answering over RDF data," in *Proc. World Wide Web, 2012*, pp. 639-648.
- [13] L. Dong, F. Wei, M. Zhou, and K. Xu, "Question answering over freebase with multi-column convolutional neural networks," in *Proc. 53rd Annu. Meet. Assoc. Comput. Linguistics 7th Int. Joint Conf. Natural Language Process. Asian Fed. Natural Language Process*, 2015, pp. 260-269.
- [14] Ren, Yuan, Artemis Parvizi, Chris Mellish, Jeff Z. Pan, Kees van Deemter and Robert Stevens. "Towards Competency Question-Driven Ontology Authoring.", *ESWC, 2014*, pp. 752-767.
- [15] X. Zhao, Z. Xing, M. A. Kabir, N. Sawada, J. Li and S. Lin, "HDSKG: Harvesting domain specific knowledge graph from content of webpages," *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), Klagenfurt, 2017*, pp. 56-67
- [16] A. K. Elmagarmid, P. G. Ipeirotis and V. S. Verykios, "Duplicate Record Detection: A Survey," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 1, 2007, pp. 1-16.
- [17] V. I. Spitzkovsky and A. X. Chang, "A cross-lingual dictionary for english wikipedia concepts," in *Proc. 8th Int. Conf. Language Resources Eval.*, 2012, pp. 3168-3175.
- [18] Suchanek F M, Kasneci G, Weikum G., "Yago: a core of semantic knowledge in Proceedings of the 16th international conference on World Wide Web", *ACM, 2007*, pp. 697-706.
- [19] Zhu, Jiangang, Haofen Wang, and Beijun Shen. "Software. zhishi. schema: A Software Programming Taxonomy Derived from Stackoverflow", In *International Semantic Web Conference (Posters & Demos)*, 2015.
- [20] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld, "Open information extraction from the web", *Communications of the ACM*, vol. 51, no. 12, pp. 68-74, 2008.

Combining Time, Keywords and Authors Information to Construct Paper Correlation Graph

Hanwen Liu, Huaizhen Kou, Xiaoxiao Chi, Lianyong Qi*

School of Information Science and Engineering
Qufu Normal University, Rizhao, China

E-mail: lianyongqi@qfnu.edu.cn

Abstract—Nowadays, recommender systems have become one of the main tools and methods for users to search for their interested papers from massive candidates. Typically, through analyzing the typed keywords by a user, a recommender system can easily retrieve the papers that cover the keywords, in an efficient and economic manner. However, one paper often only contains partial keywords that the user is interested in; therefore, the recommender system needs to analyze a pre-built paper citation graph and then return a set of papers that collectively satisfy the user's requested keywords. While the existing paper citation graph does not consider the possible self-citations and potential correlations among the papers that are not connected in the paper citation graph but with close publication time. Considering the above drawbacks, in this paper, we propose a link prediction approach that combines time, keywords and authors information for constructing a new relation graph. Finally, a case study is employed to explain our approach step by step and demonstrate the feasibility of our proposal.

Keywords—link prediction; paper citation graph; paper correlation graph; time; keywords; author information

I. INTRODUCTION

Currently, when searching for interested papers via existing paper search websites, e.g., Google Scholar and Baidu Academic, users can type their preferred keywords and then the websites will recommend appropriate papers that cover the typed keywords to the users [1]. Generally, a paper often contains only partial keywords that a user is interested in; therefore, to meet the user's paper search requirement, a paper recommender system often needs to return the user a set of papers that collectively cover all the requested keywords. However, the keywords of a paper can only represent the paper topics or themes; therefore, considering keywords only in paper search process may generate a set of papers that belong to different research domains and are actually not correlated, which fails to satisfy the original user requirements on deep and continuous research on a certain domain or topic.

Fortunately, paper citation graphs that depict the citation relationships among different papers have provided a promising way to model the paper correlations from both width and depth perspectives. However, current paper citation graphs still face a big challenge, i.e., they do not consider the possible self-citations from authors and potential correlations among the papers not connected in the paper citation graphs but with close publication time.

Considering this challenge, we propose a novel link prediction approach to improve the traditional paper citation graphs, as link prediction has already been proven the best solution for various link optimization problems in graphs [2][3]. More specifically, link prediction attempts to estimate the likelihood of the existence of a link between two nodes based on the existing properties information of nodes and network structures.

Overall, our contributions in this paper are three-fold:

- We propose a novel link prediction approach to construct new relation graphs among papers (i.e., paper correlation graphs). Our proposal considers a wide range of factors that influence the correlations among different papers, such as paper publication time, paper keywords and paper authors. In addition, our link prediction approach takes the network structure of paper citation graphs into considerations, which makes the predicted results more reasonable and convincing.
- We improve the existing paper citation graphs by reducing the negative influence of intentional self-citations from partial authors.
- At last, we evaluate the feasibility of our proposal through a case study.

The rest of paper is organized as follows. Related work is presented in Section II. In Section III, we introduce the research motivation. In Section IV, the details of our proposed link prediction approach is described. A case study is investigated in Section V to demonstrate the effectiveness of our link prediction approach. Finally, in Section VI, we summarize this paper.

II. RELATED WORK

Currently, link prediction has made massive strides in many research areas and played an important role in more and more fields. According to [4], link prediction approaches can be classified into three categories: similarity-based methods, maximum likelihood approaches and probabilistic methods. However, the similarity-based methods can be used to the large-scale networks, which is because it can calculate the similarity score between two nodes [5]; although maximum likelihood approaches can obtain specific parameters and probabilistic methods can predict missing links by using the trained model, maximum likelihood approaches and probabilistic methods often fail to deal with the large-scale networks [6]. Therefore, in our research we mainly consider the similarity-based approach. In addition, the work in [7] investigated the use of link strength

for the link prediction problem, and they proposed the weighting criterion was based absolutely on topological data: the frequency of existing interactions (i.e. the number of edges) between nodes in the social networks. But they don't take full advantage of node information in the weighting criterion.

In view of existing link prediction approaches, a novel the link prediction approach to construct the paper correlation graph, that is, the similarity-based weighting method.

III. RESEARCH MOTIVATION

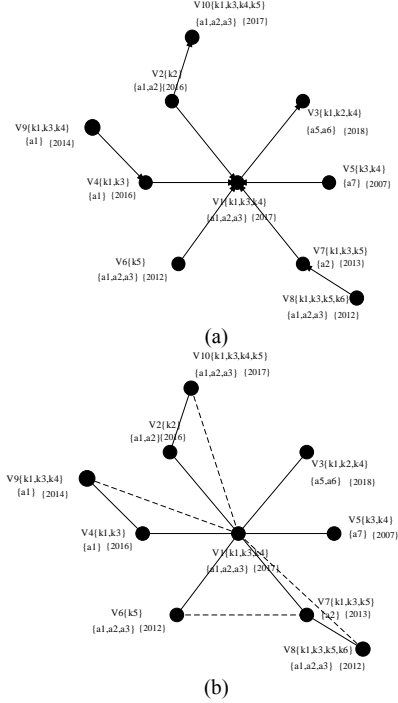


Figure 1. (a) Paper citation graph and (b) Paper correlation graph.

An intuitive example is presented in Fig.1 to motivate our paper. Assume that there is a paper citation graph G_C and a paper correlation graph G_P , Fig.1(a) and Fig.1(b) are a part of G_C and G_P , respectively. Fig.1 contains 10 nodes, i.e., v_1, \dots, v_{10} , each of which represents a paper and contains some node attributes (i.e., paper publication time, paper keywords and paper authors). In Fig.1(a), the self-citation relationship between node v_1 and node v_2 in the paper citation graph is generated merely due to the common authors of v_1 and v_2 , which is not reasonable and fair for accurate paper recommendation. Therefore, in this paper, we need to reduce the effect of the intentional self-citation phenomenon through a weighted approach. Besides, in Fig. 1(a), node v_1 and node v_{10} are published in same period and they also have common keywords and common authors, but there is no link (edge) between them. Thus, in Fig.1 (b), we need to establish the new link between node v_1 and node v_{10} by using the link prediction approach. In view of the aforementioned analyses, a link prediction approach is necessary to improve current paper citation graphs, which will be introduced in detail in Section IV.

IV. LINK PREDICTION APPROACH

According to the analysis of the research motivation, we propose a link prediction approach by using the attributes information and network structure of nodes. To the best of our knowledge, the fundamental process of the unsupervised link prediction model follows the task sequence, which was first proposed by Kleinberg [8]. Concretely, our process of link prediction approach can be seen from Fig. 2. This process mainly consists of the following five activities:

Activity 1: Pre-processing of the graph. In our research, the paper citation graph (G_C) is regarded as an undirected paper citation graph (G), which is because it is easier to construct the paper correlation graph.

Activity 2: Graph partition. In this activity, the G is divided in to two parts. One is training sub-graph (G_{train}) and another one is test sub-graph (G_{test}). In the G_{train} , we need to get the Maximum Score from existing pairs of nodes. And in the G_{test} , we need to get the weighted values of the two unconnected nodes.

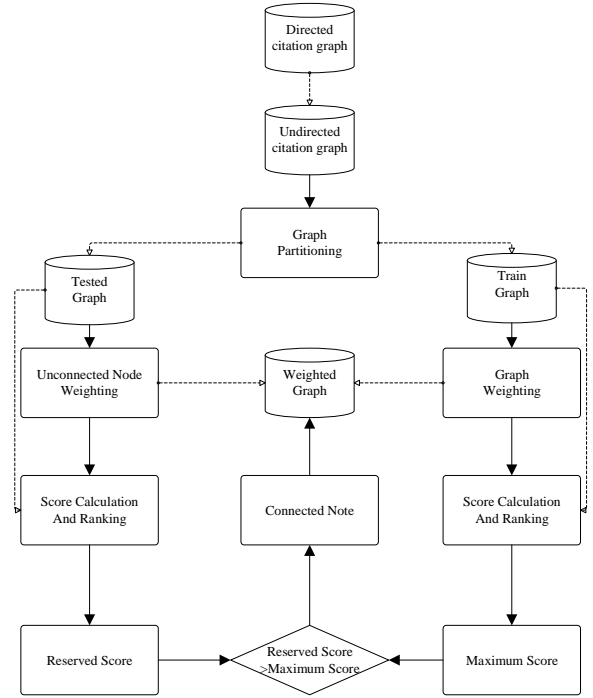


Figure 2. Process for weighting-based link prediction.

Activity 3: Graph to be weighted. The weights of the two connected nodes are calculated by using the weighting criteria in the G_{train} and the weights of two unconnected nodes are calculated in the G_{test} .

Activity 4: Score calculation and ranking. (1) Firstly, we use a similarity function formula WCN to calculate a weight value of two unconnected nodes in the G_{train} . Then we produce a ranking list in descending order of score. At last, the Maximum Score is saved in $w_{max}(v_{itrain}, v_{jtrain})$.

The Weighted Common Neighbor - $WCN(v_{itrain}, v_{jtrain})$ and the Maximum Score - $w_{max}(v_{itrain}, v_{jtrain})$:

$$\sum_{v_{jtrain} \in \Gamma(v_{itrain}) \cap \Gamma(v_{jtrain})} \frac{w(v_{itrain}, v_{jtrain}) + w(v_{jtrain}, v_{itrain})}{2} \quad (1)$$

$$w_{train}^{WCN}(v_{itrain}, v_{jtrain}) = \frac{WCN(v_{itrain}, v_{jtrain})}{|\Gamma(v_{itrain}) \cap \Gamma(v_{jtrain})|} \quad (2)$$

$$w_{max}(v_{itrain}, v_{jtrain}) = \arg \max_{i,j=1,N} w_{train}(v_{itrain}, v_{jtrain}) \quad (3)$$

Where $|\Gamma(v_{itrain}) \cap \Gamma(v_{jtrain})|$ represents the number of common neighboring nodes of node v_{itrain} and node v_{jtrain} .

(2) In the G_{test} , we will perform score calculation of two unconnected nodes and produce a descending ranking list.

Activity 5: Connecting nodes. LP (link prediction) is defined as in equation (4):

$$LP = \{w_{test}(v_{itest}, v_{jtest}) > w_{max}(v_{itrain}, v_{jtrain})\} \quad (4)$$

A. Proposed Weighting Criteria

Consider that each paper of the G contain paper attributes information (time, keywords and authors). In addition, the link prediction approach offers the similarity functions WCN that can be used for the weight calculation. Therefore, here we consider three sets of those functions: Time, Keyword and Author, and we propose the general weighting model as described in Eq. (5), where $time \in Time$, $keyword \in Keyword$, $author \in Author$ and $x_{time}, x_{keyword}, x_{author} \in \{0, 1\}$.

$$w^*(v_i, v_j) = time^{x_{time}} \times keyword^{x_{keyword}} \times author^{x_{author}} \quad (5)$$

The proposed general weighting model allows the generation of the different weighting criteria by Eq. (5). In addition, it is significant to emphasize that the product between the weighting criteria in link prediction approach formulation ensures that the selected node attributes must be considered simultaneously. Thus, we propose two different weighting criteria as below:

Keywords and Authors Weighting. In our research, if the number of common keywords and co-authors of two papers increases, the weighted values between the two nodes will be greater. But when there is no common keyword in two papers, the weighted values between the two nodes will decrease as the number of co-authors increases. Such strategies have been adopted to reduce the effect of the self-referencing. Thus, the weighting criteria for a pair of nodes v_i and v_j are defined as in equations (6)-(9):

$$\gamma = \begin{cases} 1 & \text{Contains common keywords} \\ 0 & \text{Otherwise} \end{cases} \quad (6)$$

$$cosine(K_{v_i}^a, K_{v_j}^a) = \frac{|K_{v_i}^a \cap K_{v_j}^a|}{\sqrt{|K_{v_i}^a|} \times \sqrt{|K_{v_j}^a|}} \quad (7)$$

$$cosine(A_{v_i}^a, A_{v_j}^a) = \frac{|A_{v_i}^a \cap A_{v_j}^a|}{\sqrt{|A_{v_i}^a|} \times \sqrt{|A_{v_j}^a|}} \quad (8)$$

$$w^{KA}(v_i, v_j) = C \times (r \times \beta^{(1-cosine(K_{v_i}^a, K_{v_j}^a))} \times \alpha^{(1-cosine(A_{v_i}^a, A_{v_j}^a))} + (1-r) \times \beta \times \alpha^{cosine(A_{v_i}^a, A_{v_j}^a)}) \quad (9)$$

Where α/β ($0 < \alpha, \beta < 1$) is arbitrary damping parameters used to calibrate the importance of authors and keywords in the weighting criteria. $A_{v_i}^a / A_{v_j}^a$ ($K_{v_i}^a / K_{v_j}^a$) is a set of authors (keywords) of the node v_i/v_j . $A_{v_i}^a \cap A_{v_j}^a / K_{v_i}^a \cap K_{v_j}^a$ represents the node v_i and node v_j have same authors/keywords. A constant C is defined for convenience of calculation.

Time, Keywords and Authors Weighting. According to the Keywords and Authors Weighting, if the published time of two papers are relatively close, the weighted values between the two nodes will be greater. Thus, the weighting criteria for a pair of nodes v_i and v_j are defined as in equations (10)-(11):

$$k(t_p) = \frac{t_c - \max(t_{p_{v_i}}, t_{p_{v_j}})}{t_c - \min(t_{p_{v_i}}, t_{p_{v_j}})} \quad (10)$$

$$w^{TKA}(v_i, v_j) = C \times k(t_p) \times (r \times \beta^{(1-cosine(K_{v_i}^a, K_{v_j}^a))} \times \alpha^{(1-cosine(A_{v_i}^a, A_{v_j}^a))} + (1-r) \times \beta \times \alpha^{cosine(A_{v_i}^a, A_{v_j}^a)}) \quad (11)$$

Where $t_{p_{v_i}}/t_{p_{v_j}}$ indicates the time of publication of the paper p_{v_i}/p_{v_j} , t_c is the current time.

B. Paper Correlation Graph

Definition1. Paper correlation graph: Paper correlation graph is represented by $G_p = \{V_p, E_p\}$, where V_p and E_p denotes its sets of nodes and edges respectively. In addition, the paper correlation graph is undirected. Meanwhile, for each paper, the paper correlation graph G_p has a corresponding node v , and for each of nodes pair (v_i, v_j) , the paper correlation graph contains the edge $e(v_i, v_j)$ between v_i and v_j .

V. A CASE STUDY

In this section, a case study is discussed to demonstrate the process of link prediction approach. Due to the limitation of the length of the paper, the case study only considers the first weighting criteria (i.e., the Keywords and Authors Weighting) for the link prediction task. Thus, the process of constructing the paper correlation graph is demonstrated as follows:

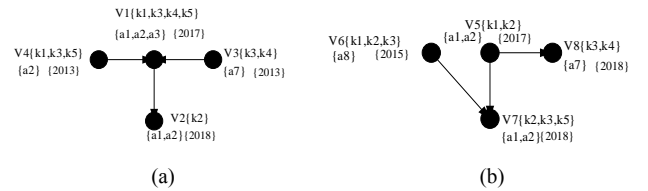


Figure 3. Paper citation graph.

Step1: Pre-processing of the graph. In our case, we regard the paper citation graph of Fig.3 as an undirected citation graph.

Step2: Graph partition. In our case, the G is divided into two parts. One is training sub-graph (G_{train}) and another one is test sub-graph (G_{test}). Therefore, in the Fig. 3, the (a) is the G_{train} and the (b) is the G_{test} .

Step3: Graph to be weighted. We use the Keywords and Authors Weighting to calculate the weight of two connected nodes in the (a) and two unconnected nodes in the (b). Meanwhile, we range the values of α ($0 < \alpha < 1$) from 0.5 to 0.7 with step 0.2, and $\beta=0.5$ (see Table 1). Here, we set up the different parameters value that will obtain the different weight value.

TABLE I. PARAMETERS SET.

Similarity function	Parameters set	
	α	β
$w^{KA}(v_i, v_j)$	0.5	0.5
$w^{KA}(v_i, v_j)$	0.7	0.5

(1) we use: $\beta=0.5, \alpha=0.5$ and $C=1$:

Weighted calculation in training sub-graph:

$$\begin{aligned} (v_1, v_2): r=0; \cosine(K_{v_1}^a, K_{v_2}^a) &= 0; \cosine(A_{v_1}^a, A_{v_2}^a) \approx 0.81; w^{KA}(v_1, v_2) \approx 0.29 \\ (v_1, v_3): r=1; \cosine(K_{v_1}^a, K_{v_3}^a) &\approx 0.71; \cosine(A_{v_1}^a, A_{v_3}^a) = 0; w^{KA}(v_1, v_3) \approx 0.41 \\ (v_1, v_4): r=1; \cosine(K_{v_1}^a, K_{v_4}^a) &\approx 0.87; \cosine(A_{v_1}^a, A_{v_4}^a) \approx 0.58; w^{KA}(v_1, v_4) \approx 0.68 \end{aligned}$$

Weighted calculation in test sub-graph:

$$\begin{aligned} (v_5, v_6): r=1; \cosine(K_{v_5}^a, K_{v_6}^a) &\approx 0.82; \cosine(A_{v_5}^a, A_{v_6}^a) = 0; w^{KA}(v_5, v_6) \approx 0.44 \\ (v_6, v_8): r=1; \cosine(K_{v_6}^a, K_{v_8}^a) &\approx 0.41; \cosine(A_{v_6}^a, A_{v_8}^a) = 0; w^{KA}(v_6, v_8) \approx 0.33 \\ (v_7, v_8): r=1; \cosine(K_{v_7}^a, K_{v_8}^a) &\approx 0.41; \cosine(A_{v_7}^a, A_{v_8}^a) = 1; w^{KA}(v_7, v_8) \approx 0.66 \end{aligned}$$

(2) we use: $\beta=0.5, \alpha=0.7$ and $C=1$:

Weighted calculation in training sub-graph:

$$\begin{aligned} (v_1, v_2): r=0; \cosine(K_{v_1}^a, K_{v_2}^a) &= 0; \cosine(A_{v_1}^a, A_{v_2}^a) \approx 0.81; w^{KA}(v_1, v_2) \approx 0.37 \\ (v_1, v_3): r=1; \cosine(K_{v_1}^a, K_{v_3}^a) &\approx 0.71; \cosine(A_{v_1}^a, A_{v_3}^a) = 0; w^{KA}(v_1, v_3) \approx 0.57 \\ (v_1, v_4): r=1; \cosine(K_{v_1}^a, K_{v_4}^a) &\approx 0.87; \cosine(A_{v_1}^a, A_{v_4}^a) \approx 0.58; w^{KA}(v_1, v_4) \approx 0.79 \end{aligned}$$

Weighted calculation in test sub-graph:

$$\begin{aligned} (v_5, v_6): r=1; \cosine(K_{v_5}^a, K_{v_6}^a) &\approx 0.82; \cosine(A_{v_5}^a, A_{v_6}^a) = 0; w^{KA}(v_5, v_6) \approx 0.62 \\ (v_6, v_8): r=1; \cosine(K_{v_6}^a, K_{v_8}^a) &\approx 0.41; \cosine(A_{v_6}^a, A_{v_8}^a) = 0; w^{KA}(v_6, v_8) \approx 0.47 \\ (v_7, v_8): r=1; \cosine(K_{v_7}^a, K_{v_8}^a) &\approx 0.41; \cosine(A_{v_7}^a, A_{v_8}^a) = 1; w^{KA}(v_7, v_8) \approx 0.66 \end{aligned}$$

Step4: Score calculation and ranking.

(1) For KA weighting criteria, when $\beta = 0.5$ and $\alpha=0.5$, we can get the Maximum Score, i.e., $w_{max}(v_{itrain}, v_{jtrain}) = w_{train}(v_2, v_4) \approx 0.55$. When $\beta = 0.5$ and $\alpha=0.7$, we can get the Maximum Score, i.e., $w_{max}(v_{itrain}, v_{jtrain}) = w_{train}(v_2, v_4) \approx 0.68$.

(2) In the G_{test} , we can get such a ranking list that $w^{KA}(v_7, v_8) > w^{KA}(v_5, v_6) > w^{KA}(v_7, v_8)$.

Step5: Connecting nodes. Seen from the Step 4, When $\beta = 0.5$ and $\alpha=0.5$, we can get such a ranking result that $w^{KA}(v_7, v_8) > w_{max}(v_{itrain}, v_{jtrain}) > w^{KA}(v_5, v_6) > w^{KA}(v_7, v_8)$. Therefore, we can draw a conclusion that v_7 with v_8 constructs a new link. And we can construct the paper correlation graph by connecting a pair of nodes v_7 with v_8 .

VI. CONCLUSIONS

In this paper, we mainly put forward a novel link prediction approach to construct the paper correlation graph. In addition, we investigated whether the combination of time, keywords and authors information in the weight computation could reduce the

effect of the self-citation. Finally, the feasibility of this the link prediction approach is validated by a case study. In the future work, we will design and deploy a set of real-world experiments to further prove the feasibility of our proposal. Besides, as recommendation process often involves the data privacy issues [9-18], we will further refine our work by considering the privacy-preservation.

ACKNOWLEDGMENT

This research is partially supported by the National Science Foundation of China (No. 61872219).

REFERENCE

- [1] L. Pan, X. Dai, S. Huang, J. Chen, Academic Paper Recommendation Based on Heterogeneous Graph. Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data. Springer. (2015), pp.381–392.
- [2] B. Yan, S. Gregory, Finding missing edges in networks based on their community structure, Phys. Rev. E 85 (5) (2012), pp. 056112-056117.
- [3] P. Wang, B. Xu, Y. Wu, X. Zhou, Link prediction in social networks: the state-of-the-art, Sci. China Inf. Sci. 58 (1), (2015), pp.1–38.
- [4] L. Lü T. Zhou. Link prediction in complex networks: a survey. Phys. A. 390 (6), (2011), pp.1150–1170.
- [5] A. Clauset, C. Moore, M.E. Newman. Hierarchical structure and the prediction of missing links in networks. Nature 453, (2008), pp.98-101.
- [6] R.H. Li, J.X. Yu, J. Liu, Link prediction: the power of maximal entropy random walk, Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11, (2011), pp.1147-1156.
- [7] C. P. Muniz, R. Goldschmidt, and R. Choren, Combining contextual, temporal and topological information for unsupervised link prediction in social networks, Knowledge-Based Syst., vol. 156, (2018), pp.129–137.
- [8] P.M. Chuan, L. H. Son, M. Ali, T. D. Khang, N. Dey, Link prediction in co-authorship networks based on hybrid content similarity metric. Applied Intelligence, 48(8), (2018), pp.2470-248.
- [9] L. Qi, X., W. Dou, Q. Ni. A Distributed Locality-Sensitive Hashing based Approach for Cloud Service Recommendation from Multi-Source Data. IEEE Journal on Selected Areas in Communications, 35(11): 2616-2624, 2017.
- [10] Y. X, H. Liu, C. Yan. A privacy-preserving exception handling approach for dynamic mobile crowdsourcing applications in EURASIP Journal on Wireless Communications and Networking, 2019, pages:113.
- [11] Y. Xu, L. Qi, W. Dou, J. Yu. Privacy-preserving and Scalable Service Recommendation based on SimHash in A Distributed Cloud Environment. Complexity, Volume 2017, Article ID 3437854, 9 pages, 2017.
- [12] L. Qi, R. Wang, S. Li, Q. He, X. Xu, C. Hu. Time-aware Distributed Service Recommendation with Privacy-preservation. Information Sciences, 480: 354-364, 2019.
- [13] L. Qi, Y. Chen, Y. Yuan, S. Fu, X. Zhang, X. Xu. A QoS-Aware Virtual Machine Scheduling Method for Energy Conservation in Cloud-based Cyber-Physical Systems. World Wide Web Journal, 2019.
- [14] W. Gong, L. Qi, Y. Xu. Privacy-aware Multidimensional Mobile Service Quality Prediction and Recommendation in Distributed Fog Environment. Wireless Communications and Mobile Computing, vol. 2018, Article ID 3075849, 8 pages, 2018.
- [15] L. Qi, W. Dou, W. Wang, G. Li, H. Yu, S. Wan. Dynamic Mobile Crowdsourcing Selection for Electricity Load Forecasting. IEEE ACCESS, 6: 46926-46937, 2018.
- [16] C. Yan, X. Cui, L. Qi, X. Xu, X. Zhang. Privacy-aware Data Publishing and Integration for Collaborative Service Recommendation. IEEE ACCESS, 6: 43021-43028, 2018.
- [17] L. Qi, X. Zhang, W. Dou, C. Hu, C. Yang, J. Chen. A Two-stage Locality-Sensitive Hashing Based Approach for Privacy-Preserving Mobile Service Recommendation in Cross-Platform Edge Environment. Future Generation Computer Systems, 88: 636-643, 2018.
- [18] L. Qi, S. Meng, X. Zhang, R. Wang, X. Xu, Z. Zhou, W. Dou. An Exception Handling Approach for Privacy-preserving Service Recommendation Failure in A Cloud Environment. Sensors, 18(7): 1-11, 2018.

Modeling and Simulation of CPS based on SysML and Modelica(KG)

Fei Deng, Yunqiang Yan, Feng Gao, Linbo Wu
Institute of Computer Application
China Academy of Engineering Physics
MianYang, China
fax_caep@163.com

Abstract—Cyber-Physical Systems (CPS) is usually associated with large numbers of domains. The domain relevance allows system engineers' knowledge to spread many domains. The efficiency and accuracy of modeling are not able to be guaranteed. Therefore, in this paper, a set of CPS modeling guideline is proposed based on SysML, and the system's 4-layer abstract hierarchy and the kind of modeling products obtained on each layer are defined. Based on this, a modeling specification containing seven sub-processes is designed. In order to verify the correctness of CPS function and the consistency of the business logic at the primary phase of the design, we summarize the mapping rules of SysML-Modelica and define the algorithm of model conversion. Finally, a simple CPS case is used to verify our task. The results show that this method can be effective in CPS's modeling and Simulation.

Keywords—Cyber-Physical Systems; Modeling ;SysML; Modelica;

I. INTRODUCTION

CPS is organic and in-depth integrity of Computation, Communication, and Control technologies, and a next-generation intelligent system that closely integrates and coordinates computational resources with physical resources [1-3]. CPS integrates such systems engineering as environmental perception, embedded computing, and network communication, closely coordinates computing and physical resources, covering all aspects of social life. As computing technology, network technology and control technology are constantly developing, CPS has become a new trend of research and development of modern information technology. Modeling and simulation are quite significant to the construction of CPS. It can not only verify the CPS at the primary phase of the design but also is an important section of model-based development and testing.

Since CPS is usually associated with multiple domains and the domain relevance allows the system engineers' knowledge to associate with many domains [4]. The efficiency and accuracy of modeling are not able to be guaranteed. Therefore, a set of the modeling guide of the CPS system based on SysML is proposed in this paper, and 4-layer abstract hierarchy of the system and the kind of modeling products obtained on each layer are defined. Based on this, a modeling specification containing seven sub-processes is designed. In order to verify the correctness of CPS function and the consistency of business logic at the primary phase of design, we summarize the mapping rules of SysML to Modelica model and define the algorithm of model conversion. Finally, a simple temperature control system case is used to verify our method.

The structure of this paper is organized as follows: In the DOI reference number: 10.18293/SEKE2019-167

second section, the background and the related work are overviewed. In the third section, the overall framework is proposed, and CPS modeling method, model mapping cabinet and conversion algorithm are described in detail. In the fourth section, the case analysis is performed. In the fifth section, summary and prospects are made.

II. BACKGROUND AND RELATED WORK

A. Cyber-Physical System

The typical CPS architecture mainly includes the following three kinds of parts: sensor, actuator, and controller [6]. The sensor is used to perceive all information of the physical domain. The controller can accept information from the sensor and send orders according to the control logic. The actuator receives a control order and begins to control the physical objects. The operation mechanism among the basic components is shown in Fig. 1.

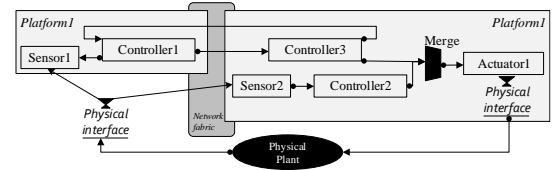


Fig. 1. C typical Cyber-physical system

In the simulation study of the CPS system, Lin Jing et al. from the Missouri University of Technology used agent-based modeling to construct a model of CPS [7]. Based on the service-oriented architecture (SOA), the University of Texas proposed Physical-entity service model for CPS modeling [8]. Frank Wawrzik et al. proposed a SysML-based CPS modeling simulation method SICYPHOS CPS [4].

SysML (System Modeling Language) is a system modeling language extended from UML (United Modeling Language) [5,9-10]. Based on the nine types of SysML graphs, system engineering personnel can easily regulate term, model, design, and analysis and verification on the system [11], which is widely used in the industry field. Although SysML is widely used now, from the perspective of system security analysis, the current SysML specification cannot effectively support the dynamic simulation of physical engineering systems. Therefore, Modelica[12-13] will be selected for dynamic simulation of CPS in this paper.

III. MODELING AND SIMULATION OF CPS

Based on the analysis of SysML and Modelica features and the current challenges for CPS simulation and verification, we designed a CPS modeling and simulation framework based on

SysML and Modelica, and realized the SysML-Modelica automatic conversion tool (Fig.3).

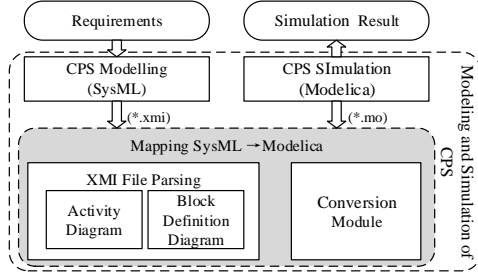


Fig. 2. Logic framework diagram of conversion

The overall logical framework is shown in Fig. 2: Firstly, construct SysML model according to the files and requirements of system design, and export of XMI [14] data model, and the automatic conversion of SysML-Modelica is realized according to the mapping rules of the construction model. Finally, input the Modelica model to verify the correctness of CPS function and the consistency of the business logic.

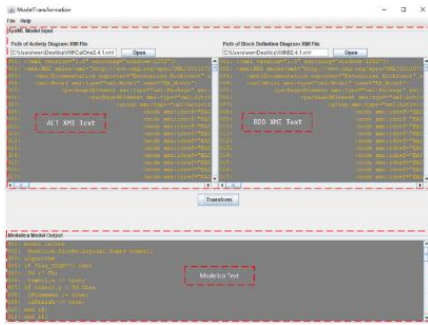


Fig. 3. Model Conversion Tool GUI

The workflow of data conversion processing and conversation in this process is shown in Fig. 4:

- 1) Use Enterprise Design (EA) to construct the CPS behavior model and structural model, and use export function owned by the EA tool to export the CPS SysML model as the XMI file;
- 2) Analyze XMI files with Java-based extension tool DOM4J;
- 3) Convert the extracted information into the grammatical format conforming to the Modelica modeling language, and output Modelica Text semantically equivalent to SysML information;
- 4) Use OpenModelica to simulate Modelica Text.

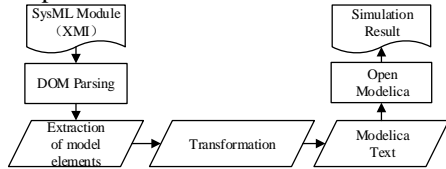


Fig. 4. Flow of data processing

There are two main technical points to achieving this work: (1) CPS modeling: Due to the increasing complexity of CPS, it makes the system modeling process more complex and difficult; (2) The conversion rules of SysML-Modelica modeling languages: Since SysML and Modelica are two different modeling languages, there are differences in

grammatical descriptions in many model elements. Therefore, we need to construct a set of semantic equivalent model conversion rules based on model semantics.

A. CPS Modeling

From the above analysis on the structure of the CPS system, it can be clearly known that all physical entities in the CPS only belong to one of these three types. When modeling the static structure, a structural model can be constructed for each entity. After structural modeling, it is required to model the dynamic behavior of each entity. When the industry world constructs the corresponding SysML model, select Block Definition Diagram (BDD) and Internal Block Diagram (IBD) that can express system architecture information, to construct a structure model and use Activity Diagram (ACT), Sequence Diagram (SD), or State Machine Diagram (STM) to express dynamic activity information. Considering the semantic features contained in our Modelica model, mainly use BDD to construct the structure model of CPS and use ACT to construct the activity model of CPS in this paper.

a) Modeling Guidelines

The complex structure and activity of CPS allow the modeling process of the system to be more complicated and difficult to grasp. One of the most effective ways to solving these problems is to construct a hierarchical model for the system. Abstraction layers that are hierarchical and clearly defined can effectively reduce system complexity which helps system modeling personnel to create and manage system models at different levels of granularity. We define the system 4-layer abstract hierarchy, which is the system layer, function layer, software and hardware layer, and deployment layer from top to bottom with an increasing amount of granularity described in each layer.

In the practical modeling, from the primary requirement of CPS, the modeling design process is divided into 7 sub-processes based on the four levels, and in each modeling process, different modeling tasks need completing.

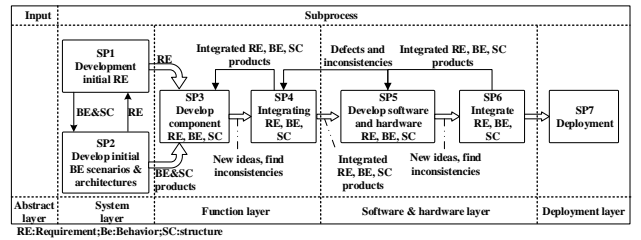


Fig. 5. Modeling process

1) *Sub-process SP1* supports the development of primary requirements for system layer. The top-level requirements of the system are described from the angel of system realization according to the system prospect and the needs of relevant stakeholders.

2) *Sub-process SP2* supports system layer initial activity scenarios and the development of system structure. The initial behavior scenario describes a coarse- granular workflow; The development of the initial architecture mainly includes limiting system boundaries, defining system interfaces, and linkage between systems and external environment entities.

3) *Sub-process SP3*: Based on the output of SP1 and SP2, refine and expand the initial requirements, activity scenarios, and architecture, and focus point shifts to the internal system.

This process is a constant iterative course until the system engineering gets a satisfactory result.

4) *Sub-process SP4* is responsible for coordinating and integrating the cross-system layer and functional layer as well as the inconsistencies of requirements, activities, and structural products.

5) *Sub-process SP5* develops the requirements, activities, and architecture of the hardware and software layer based on the results of SP4 integration. The focus shifts from logical functional components to hardware or software components.

6) *Sub-process SP6* is similar to sub-process SP4, in charge of coordinating and integrating across functional layer and hardware layer as well as inconsistencies in requirements, activities, and structural products within the software layer.

7) *Sub-process SP7* is responsible for the designing scheme of deploying software and hardware to physical units.

B. XMI file Parsing

The information in the diagram can be read by parsing the XMI file of the active graph and the module definition graph.

The block definition diagram displays the static structure information of the system. Its main element is block. We can obtain the information contained in the entity through parsing block and its value attribute tags, and use Variable and Parameter respectively to save them. The Variable data type is used to store variables information in a block. As shown in Table 1, it includes two attributes: name and type, the name represents the variable name, and type represents the variable type. The Parameter data type stores information of parameters in a block, containing attributes' type, name, and value. The attributes' name and type have the same meaning as the Variable data type, and the value attribute represents the default value of the parameter.

TABLE I. THE MAPPING RELATIONSHIP BETWEEN XMI TAGS AND GRAPH ELEMENTS

Model	Graph element	XMI Label	Define data types	Contains properties
BDD	block	<packagedElement xmi:type="uml:Class">	Variable	type, name
	Value attribute	<ownedAttribute>	Parameter	type, name, value
	Activity zoning	<group>	ActivityPartition	name, classname, id
ACT	node	<node> <ownedParameter>	Node	id, name, classname, owner, type
	Transfer edge	<edge> <guard>	Transition	id, source, target, guard

The main elements in the activity diagram include nodes, edges, detection values of the edge, activity parameters, and active partitions. The basic element information corresponds to node, edge, guard, ownedParameter, and group of the label in the XMI file. As shown in Table 1, we store the parsed active graph elements into the following three data types: Node, Transition, and ActivityPartition. The Node data stores information of the control nodes, action nodes, object nodes, and active parameter elements in the SysML activity diagram, which contains 5 attributes in total: ID name, classname, owner, and type, respectively. Among them, type represents the node type and contains owned Parameter. The Transition data type stores information of the edge elements in the SysML activity diagram. Its specific type is shown as follow and contains 4 attributes in total like ID, source, target, and guard. id uniquely identifies one edge, source and target are the id values of the edge source node and the target node, and guard is the monitor value, that is, the transfer condition of the edge. The ActivityPartition data type stores information of active partition elements in the SysML activity diagram and contains three attributes in total like id, name, and classname. Id uniquely identifies an active partition, name is the name of the active partition, and classname is the type of active partition.

Based on the above analysis of the block definition graph and activity graph XMI file and customized data types, the element information under the tags is stored according to the corresponding data type created by traversing each label in the XMI file as required.

C. Mapping Rule of SysML2Modelica

According to the modeling rules for CPS in the previous section, we will obtain a series of block definition diagrams and activity diagrams. After establishing the mapping rule,

SysML model can be converted to Modelica model. The element in the block definition diagram is mainly converted to an element in the Modelica model declaration area, and the elements in the activity diagram are mainly converted to elements in the Algorithm area of Modelica model.

The block of the block definition diagram is the basic unit in SysML and corresponds to an instance of the Modelica model, thus establishing the mapping rules for SysML block and Modelica model. The value property of the block maps the variables of Modelica model; The port and components of the block (Instances of other modules) map to the member type in Modelica model. They are directly mapped to Modelica model equation if modular constraints are not combined with other SysML activity diagrams. At this, the structure of the module has been basically built successfully in Modelica model.

When modeling CPS, we use activity charts to represent the activity of CPS. The action in the activity diagram represents processing or transformation, so it is mapped to the equation of Modelica. Determines that the node is mapped to if-else judgment, and the merge node indicates that the above input continues to execute when arriving, and is mapped to continue executing downwards in the code of Modelica model.

The following table summarizes the rules for mapping elements in the above two SysML diagrams to Modelica elements.

TABLE II. MAPPING RULES BETWEEN SYSML AND MODELICA

	SysML	Modelica
BDD	block	model
	interfaceblock	connector
	port node	instance of
	value	connector variable

	constraint	equation
	decision node	if-else
ACT	merge node	execute sequentially
	action	equation
	State	Step
	Transition(no trigger)	Transiton

According to the above mapping rules, the model conversion algorithm is defined according to section 3.4, and the SysML diagram can be directly converted to Modelica code.

D. Model Conversion Algorithm

The conversion algorithm converts the block definition diagram and activity diagram into the Modelica model according to the conversion rules. The input of the algorithm is the XMI file for the block definition diagram and the active graph, and the output is Modelica model. The basic idea as follows: Firstly, use the resolution algorithm of the active graph XMI file to obtain all active partition collections, node collections, and edge collections. Construct a Modelica model for each active partition, and use the XMI file resolution algorithm in block definition graph to obtain the corresponding block variable collection and parameter collection, and then declare these variables and parameters in the model declaration area. If the activity diagram contains latency actions for relative time events, Declare an instance of Timer in the model declaration area. Then, go down from the starting node of the active partition and output different contents in the model algorithm region according to the node type, including selection, circulation, and concurrent structure processing. The specific model conversion algorithm is shown in Table 3 of the arithmetic.

TABLE III. SysML-MODELICA CONVERSION ALGORITHM

Algorithm: SysML2Modelica	
Input: Activity zoning set AP, node set N, edge set T, variable set V, parameter set P	
Output: Modelica model	
Procedure SysMLtransformModelica(M) Begin	
1. for each ap in AP do	
2. Create Modelica Model;	
3. Export variables and parameter declarations based on V and P;	
5. Get the set of all waiting time action nodes: TimerNodes;	
6. for each TimerNode in TimerNodes	
7. Declare an instance of ModelicaTimer;	
8. end for	
10. node= Initial node of activity zoning;	
11. while(node!=null) do	
12. if(node=="Action") then output"node.name";	
14. else if(node=="Decision") then output "if + node.name + then";	
16. else if(node=="AcceptEvent") then output "if + node.name + then";	
18. else if(node=="AcceptEventTimer") then	
19. if Relative time events then output "if time > Relative time then";	
21. else output "when time > Absolute time then";	
23. end if	
24. else if(node=="Merge") then	
25. if Merge nodes and decision nodes form a cycle then	
26. output "while +Decision node name Attribute value+ loop";	
27. else output "end if;"	
29. end if	
30. else if(node=="Fork") then	
31. Each concurrent branch continues to call the transformation algorithm;	
32. else if(node=="ActivityFinal" or "FlowFinal") then	
33. output End statement that match statements if- when	
34. break;	
35. end if	
36. Find the next node of node;	

```

37.   node=nextNode;
38. end while
39.end for
End Procedure SysML2Modelica

```

IV. CASE

In this section, a simple temperature control system [15] is used to illustrate how to model CPS, to convert into the Modelica model by mapping rules and finally conduct stimulation. The temperature control system is a temperature adjustment system involving temperature sensors, air conditioners and switch controllers. The system requires temperature control between 16 °C and 28 °C and limits the season to be summer. The temperature sensor senses room temperature and transmits the temperature to the switch controller. The controller learns the temperature and sends the coolOn or coolOff order after judging. The air conditioner implements cooling operations according to the corresponding order. If the air conditioning does not work, the temperature will gradually increase as time. Due to the space limitation, in this paper mainly take the switch controller as an example to illustrate.

A. Structure Modeling

The structure analysis on the temperature control system shows that temperature sensor corresponds to sensor entity in CPS, air conditioner corresponds to the actuator entity, and controller corresponds to the controller entity. For each such entity, block is used to define the basic information in the block definition diagram. The temperature sensor has 2 ports with one port perceiving temperature and one port transmitting temperature information to the switch controller. In this case, we require the temperature sensor to learn the temperature directly from the temperature port of the air conditioner. The controller also has two ports, one of which learns the temperature from the temperature sensor as described above, and one port sends order to the air-conditioning. Similarly, the air conditioner has a port for receiving order and a port for transmitting temperature. Besides, air conditioners accept different order and follow different temperature variable equation constraint. Temperature sensors follow only one working equation constraint. At this point, the block definition diagram for each entity and that for the port can be given. The BDD of the system is shown in Fig. 6.

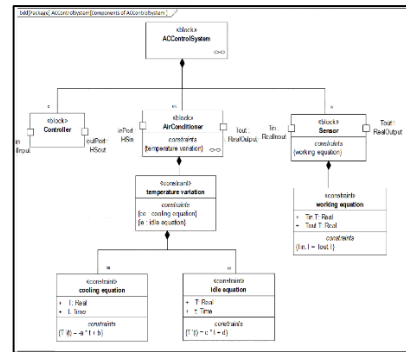


Fig. 6. System Entity Block Definition Chart

B. Behavior modeling

As mentioned above, for controllers SysML activity graph is used to represent activity information, controllers usually have complex control logic. The biggest advantage of activity graph is that they can represent the logical information well.

Fig. 7 is the activity diagram of the switch controller in the temperature control system.

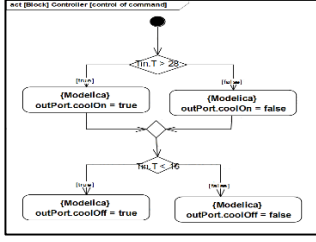


Fig. 7. Activity Chart of Switching Controller

C. SysML-Modelica Model Conversion

Firstly, according to the entity block definition diagram, we know that there are three different entities in the entire system, and each entity corresponds to Modelica model. In terms of the switch controller, combined with the block definition diagram and the activity diagram, according to the SysML-Modelica model mapping rule in Section 3.3, Modelica code shown in Fig. 8 (a) can be obtained.

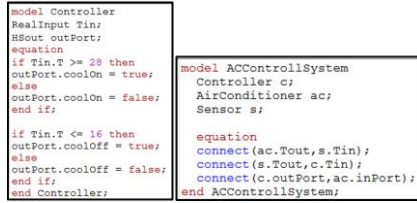


Fig. 8. Modelica code Code for Switch Controllers and Systems

Though temperature sensors and air conditioners belong to different types of CPS entities, alike switch controllers, they can be converted to Modelica code combined with module diagrams and model mapping rules. For all models required for StateGraphics library, automatically a line of Modelica code is required to "addinner Modelica.StateGraph.StateGraphRootstateGraphRoot;". It is pointed out particularly that the two temperature variance equations are quadratic equations of time t . After the time T is derived, two Timer types are required to add to the Modelica code to calculate the temperature variable. Finally, the important step is to connect all Modelica models according to BDD to form the entire temperature control system. From Fig. 6, we can see the system model Modelica code as shown in Fig. 8.

D. Analysis of Simulation Results

Introduce all model converted code into the OpenModel tool for simulation. We can know the variation of room temperature with the time in the temperature control system, as shown in Fig. 9. The entire room temperature fluctuates directly from 16 °C to 28 °C according to the switch controller.

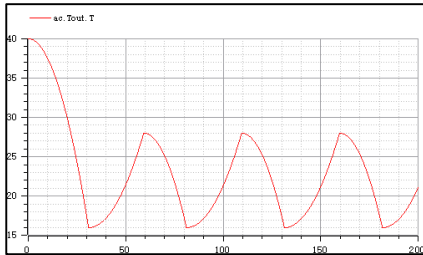


Fig. 9. Temperature variable curve

V. SUMMARY AND DISCUSS

In this paper, firstly the architecture of CPS is analyzed and divide it into three types: sensors, actuators, and controllers. For the two aspects of structure and behavior, CPS is hierarchical modeling with SysML. In order to be able to simulate modeling, the mapping rules between the SysML-Modelica models are summarized. On this basis, the model conversion algorithm is designed, the SysML-Modelica model automatic conversion tool is realized, and an case analysis is conducted by the temperature control system. Although in this article the CPS modeling and simulation process based on SysML and Modelica has been realized, there are still shortcomings. We will further study the following two aspects: 1) Extend SysML-Modelica mapping rules to support automatic conversion of more SysML-Modelica models; 2) Provide more modeling language interfaces, such as SystemC. Establish a unified modeling and simulation framework for CPS that can be applied to more domains.

REFERENCES

- [1] Lv, C., et al. "Simultaneous Observation of Hybrid States for Cyber-Physical Systems: A Case Study of Electric Vehicle Powertrain." IEEE Transactions on Cybernetics 48.8(2018):1-11.
- [2] Zanero, Stefano. "Cyber-Physical Systems."IEEE Computer50.4 (2017): 14-16.
- [3] Mo, Haining, Neeti Sharad Wagle, and Michael Zuba. "Cyber-physical systems."ACM Crossroads Student Magazine20.3 (2014): 8-9.
- [4] Wawrzik F, Chipman W, Molina J M, et al. Modeling and simulation of Cyber-Physical Systems with SICYPHOS[C] International Conference on Design & Technology of Integrated Systems in Nanoscale Era. IEEE, 2015.
- [5] OMG, "Systems Modeling Language (SysML) specification," OMG standards, formal/2013-06-01
- [6] Modelica by Example. <http://book.xogeny.com/>
- [7] JLin J, Sedigh S, Miller A. A General Framework for Quantitative Modeling of Dependability in Cyber-Physical Systems: A Proposal for Doctoral Research[J]. 2009, 1:668-671.
- [8] Huang J, Bastani F, Yen I L, et al. Extending service model to build an effective service composition framework for cyber-physical systems[C]// IEEE International Conference on Service-Oriented Computing and Applications. 2009:1-8.
- [9] Friedenthal S, Moore A, Steiner R. A Practical Guide to SysML[J]. San Francisco Jung Institute Library Journal,2013, 17(1):41-46.
- [10] Delligatti, Lenny. SysML Distilled: A Brief Guide to the Systems Modeling Language. 2013.
- [11] Weikens T. Systems engineering with SysML/UML: modeling, analysis, design[M]. Morgan Kaufmann, 2011
- [12] Fritzson P. Principles of object-oriented modeling and simulation with Modelica 2.1[M]. John Wiley & Sons, 2010.
- [13] Introduction to physical modeling with Modelica[M]. Springer Science & Business Media, 2012.
- [14] Kovse J, Härder T. Generic XMI-based UML model transformations[C]//International Conference on Object-Oriented Information Systems. Springer, Berlin, Heidelberg, 2002: 192-198.
- [15] Chen X, Ye R, Sun H, et al. Deriving Requirements Specification with Time: A Software Environment Ontology Based Approach[J]. 2013:431-43

AgileCritPath: Identifying Critical Tasks in Agile Environments

Rachel Vital PESC/COPPE Universidade Federal do Rio de Janeiro Rio de Janeiro, Brazil rachelvital@cos.ufrj.br	Glaucia Melo David R. Cheriton School of Computer Science University of Waterloo Waterloo, Canada gmelo@uwaterloo.ca	Toacy Oliveira PESC/COPPE Universidade Federal do Rio de Janeiro Rio de Janeiro, Brazil toacy@cos.ufrj.br	Paulo Alencar David R. Cheriton School of Computer Science University of Waterloo Waterloo, Canada palencar@uwaterloo.ca	Don Cowan David R. Cheriton School of Computer Science University of Waterloo Waterloo, Canada dcowan@uwaterloo.ca
--	--	--	--	--

Abstract—Planning and monitoring the execution of software development activities in agile environments are not trivial procedures. One of the main flaws of agile planning is not considering the dependencies that exist between project tasks. Dependencies between tasks found in software development project plans may lead to the emergence of critical paths, where tasks need to be handled in a strict sequence because the completion of some tasks depends on the completion of others. Not managing such critical paths may reduce team performance and delay product delivery. We performed a study to demonstrate that not identifying dependencies may impair team performance and even increase the risk of costly project delays. The study is divided into three parts: (1) an exploratory study performed in the industry; (2) the development of a tool, AgileCritPath, as a way to support development teams in identifying critical project tasks; and (3) an *in vivo* evaluation of AgileCritPath based on the Technology Acceptance Model (TAM). The results of the exploratory study provided empirical evidence that there is a need to identify and control dependencies between the tasks in the development of software in agile environments. Using the AgileCritPath tool, allowed us to introduce the Critical Path Method concepts in an agile software development organization. Moreover, the *in vivo* evaluation demonstrated the benefits of managing tasks dependencies.

Index Terms—Software Engineering, Critical Path Method, Agile, Software Development.

I. INTRODUCTION

The software development principles behind the “Manifesto for Agile Software Development” are widely used in software development project management. Although the adoption of such agile principles brings numerous benefits, including the delivery of software products on time and budget [1], estimating and planning projects using agile methodologies is still a complex process. Part of this complexity comes from the difficulty of identifying dependencies between tasks, which is a critical factor for coping with agile project failures [2].

Dependencies between tasks often lead to decreases in team’s agility level [3] and can adversely impact the delivery time of software products. For Bick and his colleagues [4], managing task dependencies is a fundamental issue in agile software development, because it can be used as a basis to define the coordination between project tasks. Additionally,

proper identification of task dependencies is important to maximize project efficiency and reduce risks [5], [6], [7], [8], [9], [10].

The critical need to manage dependencies found in agile-based software development projects is a topic that has been already explored in the literature [5], [11], [4], [12], but, currently, empirical studies on the subject are still scarce. There is a clear need to investigate this topic further, given that the lack of knowledge about task dependencies may have consequences for project duration, coordination, risk, and efficiency.

To understand the problem of identifying the dependencies properly between agile projects and software development tasks, we conducted an exploratory study in a software development organization. Through this study, we obtained empirical evidence that this is a real problem that can negatively affect software development organizations, especially when they use agile methods. After performing the exploratory study, we used the techniques of the Critical Path Method (CPM) to inform a team in a specific software development organization that already uses agile methods about the critical tasks. We have also developed a tool, which we call AgileCritPath, to evaluate the adoption of CPM in an organization. The evaluation followed the Technology Acceptance Model (TAM) [13] approach.

This paper is structured as follows. After a brief introduction in Section I, Section II describes supporting concepts. Section III presents an Exploratory Study. Section IV presents the AgileCritPath tool and a TAM evaluation of this tool. Finally, Section VI presents conclusions and future work.

II. BACKGROUND

The increased adoption of agile practices has caused traditional project management to be redefined.

Many agile methods provide some level of management for tasks, but out of the main techniques applied [14] none guarantees that teams follow what was planned. Cohn [2], in his book “Agile Estimating and Planning”, addresses the main flaws of agile planning. Among them, the author cites that tasks are not independent and that it is an error for the agile

teams to plan the tasks as if they were separate, with no need for task coordination.

In agile methods, the technical work of the development team is defined through tasks. Each team estimates tasks and, generally, they represent a small part of the expected work [2]. Tasks can be visualized using the Kanban board presented in Figure 1. Kanban boards visually depict work at various stages of a process using cards to represent tasks and columns to represent each stage of the process. Cards are moved from left to right to show progress and to help coordinate the teams performing the work. The Kanban board is a visual approach that teams use to monitor task execution.

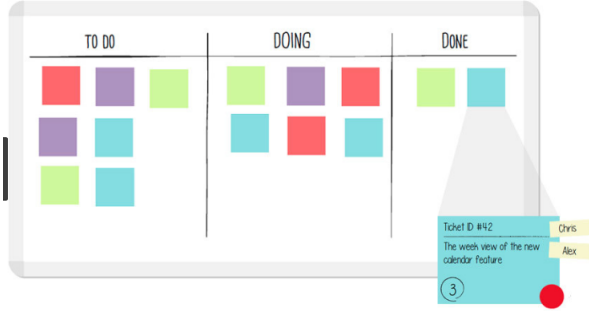


Fig. 1. Kanban board example.

Progress monitoring is an essential activity in Software management, whereby it ensures that a project plan advances according to budget, schedule, and quality expectations [15], [16]. The implementation of progress monitoring mechanisms in an agile environment is fundamental to the success of the project.

Although the Kanban framework is widely adopted in organizations, it does not show task dependencies. It is not possible to see on the Kanban board which tasks can block the execution of others and, for this reason, the flow of task execution becomes unclear to the team.

The Agile Kanban method lacks a mechanism for progress tracking. Thus, it needs to be integrated with other methods because it does not have a standard definition for software development and its specific practices have not yet been rigorously defined [12].

However, although managers can have, based on the Kanban board, support to understand progress status and progress better, they can not get information about the impact of the execution of the flow of tasks, especially when it comes to task dependencies.

III. EXPLORATORY STUDY - IDENTIFYING CRITICAL TASKS

To correctly understand the dynamics of the organization concerning project planning in an agile environment, we conducted an exploratory study in a specific software development organization. Exploratory studies have proved to be adequate in software engineering to study new ideas [17].

Through the exploratory study, we have: (1) observed the existence of dependencies between tasks in real software

development projects that use agile methods; (2) applied concepts based on the Critical Path Method to identify critical tasks in projects that use agile methods; (3) collected, analyzed and discussed the results. The exploratory study is divided into the following steps: definition of the study, data collection, data analysis, and presentation of results.

A. Definition

The study was conducted in a Brazilian software development company that has a staff of 30 employees, most of whom are software developers. The organization has been using agile methods for at least 10 years. Teams are small, usually groups of 3 to 8 people.

B. Data collection

The task execution log has been extracted from the Redmine¹ project management tool. Through the organization's task management system, we were able to identify which tasks were planned and which tasks were performed. We retrieved data from three iterations of the same project for analysis.

C. Data analysis

In this step, we identified the dependencies between tasks. The dependency identification was performed with the help of an experienced developer who was familiar with the scope of the tasks, the processes of the organization, and knew the technological architecture used in the project. Task dependencies were classified into Process and Context dependencies, following a classification suggested in a taxonomy of dependencies [5].

D. Results and discussion

From the analysis of the data, the projection of the maximum effort rate per period was obtained. Therefore, regardless of the number of people on the team, we can not plan activities with effort higher than the maximum effort rate. For the calculation of the maximum effort rate, we consider the estimated effort of the tasks performed by the team and their respective dependencies.

Figure 2 illustrates the planning performed by the team (blue line) and the maximum effort rate of the tasks (red line). The red line represents the shortest time to complete the activities and indicates the maximum rate of production (speed) at which the team can work. Regardless of the number of people, the tasks will not be completed before this deadline (red line). In this case, we can see that the team planned a delivery that can not be performed on the expected date.

Table I presents the number of dependencies identified in the study and the number of tasks defined by the team, which are extracted through the task execution log. The number of dependencies is higher than the number of tasks, within the three analyzed iterations.

¹redmine.org

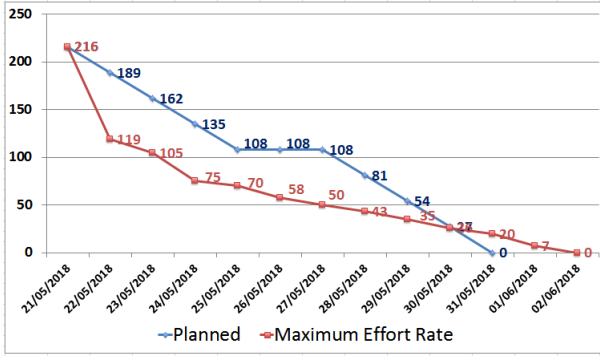


Fig. 2. The burndown graph with maximum effort rate.

TABLE I
PERCEIVED DEPENDENCIES.

Release	Task	Dep	% Dep	Process Dep	Business Dep
Release 2.12	100	96	96%	52	44
Release 2.13	69	77	112%	53	24
Release 2.14	70	84	120%	67	17

Our results show that handling task dependencies may not be straightforward when there are many tasks. Another factor we noticed was that there were several tasks, and consequently, dependencies, that are were not planned but were identified during the iteration execution.

During the analysis of the Iteration execution log, we verified that some tasks were discovered during the execution of the iteration and, consequently, the execution flow of the tasks changed because the new tasks had dependencies that should have been considered. This shows that the identification of critical tasks should occur throughout an iteration, and not only at the end.

Based on the exploratory study we conclude that there is evidence that when considering task dependencies it is possible to find the critical tasks, which if delayed, can have serious consequences, including the product release delays. Because of this evidence, we decided to develop the work further and implement AgileCritPath, a tool that could help developers identify critical tasks. The tool implementation and details are presented in Section IV.

IV. THE AGILECRITPATH TOOL

The AgileCritPath tool implements the Critical Path Method to be used in agile environments and support teams to identify and prioritize critical tasks that can block the execution of other tasks and cause costly project delays.

A. Identifying Critical Tasks

During the exploratory study, we identified that some tasks compromise or block the flow of execution of other tasks. These tasks are referred to as Critical Tasks because they can decrease the agility level of the team.

The critical path is the longest duration path through the network. The tasks that lie on the critical path cannot be delayed

without delaying the release. In agile environments, Critical Tasks are identified by building a network diagram of tasks. This network diagram is built to represent task dependencies. Figure 3 presents an example of a network diagram with seven tasks. The solid lines represent dependencies between tasks, and the dashed lines represent the association of the tasks from the beginning to the end of the graph. Critical tasks are identified as the tasks that make up the longest delivery path.

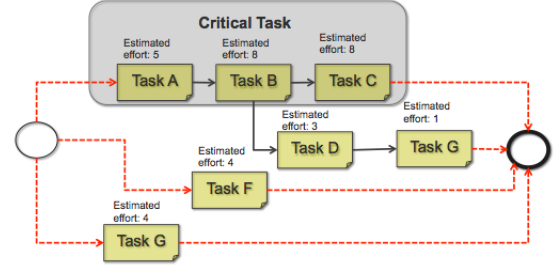


Fig. 3. Network Diagram with Critical Tasks.

B. Proposed Model

Figure 4 illustrates in detail the mechanism used to find out the Critical Task in GitHub projects.

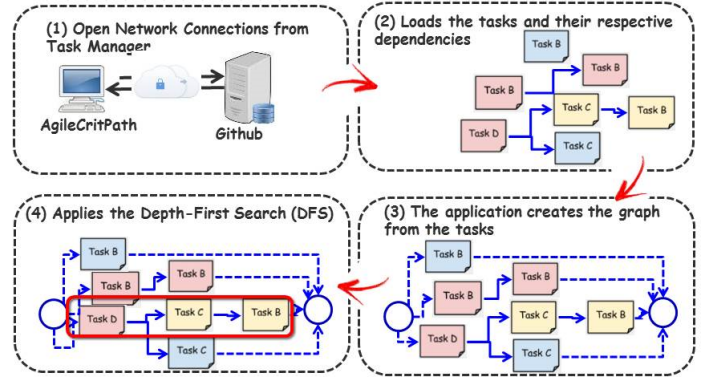


Fig. 4. Illustration of AgileCritPath.

In step (1), the application connects to the task manager to load all tasks and their dependencies. In step (2), the associations between tasks are created according to the dependencies. In step (3), the resulting graph is created. Tasks without predecessors are automatically linked to the initial node of the graph. The tasks without successors are automatically linked to the final node of the graph. In step (4), the Depth-First Search (DFS) algorithm is applied to find all paths in the graph. While reading the path, the total effort to complete the task flow is calculated.

All the found paths are displayed in descending order, from the longest path to the shortest path, according to the calculated effort for each path. Next to the path listing is the status of the task, the developer responsible for the execution of the task, and the effort of each task that makes up the path. The path size is calculated by the sum of the planned effort

reported in the individual tasks that are part of the path. If the task is finished, we consider the effort to complete the task.

Knowing all the paths in the network task is important because we can identify the “Near-Critical Paths”. Other important paths through are considered the “Near-Critical Paths” if they are at risk of becoming the Critical Path. To make this information visible to the team, we display all the paths found in the task network.

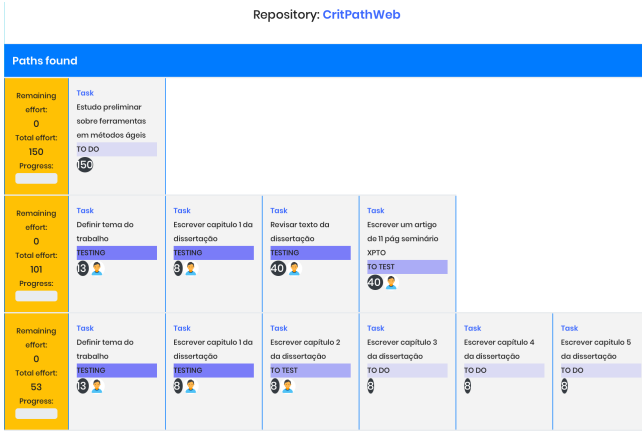


Fig. 5. Query results.

Figure 5 presents the result of querying the paths of a task network from a GitHub repository.

C. Architecture model

AgileCritPath was developed following the architecture presented in Figure 6.

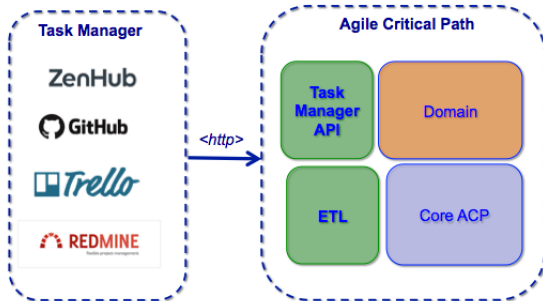


Fig. 6. The AgileCritPath architecture.

The application works autonomously and connects to the task manager through an integration API (task manager API). The Extract, Transform and Load (ETL) module performs the necessary information processing, according to the information available in the task manager. An ETL module was developed to meet the integration requirements with GitHub, ZenHub and Redmine. The ETL loads objects from the application domain, which works in isolation from the task management tools. The Core ACP module loads a task-oriented graph and implements a Depth-First Search algorithm that searches for all paths in the task network.

Planning in an agile environment is different from approaches used in traditional plan-oriented software development models [18]. Rather than employing plans in projects based on a set of predefined factors and constraints, agile models rely on the human factor to self-organize. In this work, we seek to offer systemic support, allowing the information to be accessible to all team members, and the decisions about the order of task execution to be made at any time.

V. THE AGILECRITPATH TOOL EVALUATION

The evaluation of the AgileCritPath tool was performed according to the Technology Acceptance Model (TAM). The TAM approach was proposed by Davis in 1989 [19] and suggests the acceptance of a new IT technology depending on two variables: (1) perceived ease of use and (2) perceived utility usefulness. For Davis, people tend to use or not use technology to improve their performance at work - perceived utility. However, even if a person understands that a particular technology is useful, its use may be impaired if the application is too complicated, so effort does not compensate for use - perceived ease.

A. Procedures

The general objective of using TAM was to evaluate the potential of the AgileCritPath tool in an industrial environment. The evaluation was performed in a company that has been working in the area of software engineering for 20 years and adopts development processes based on agile practices. The project used in the evaluation was selected by the organization. The selected company uses Redmine as a task management tool. For the execution of the study, we modified our tool so it could access Redmine.

B. Execution

The study was performed in-vivo. *In-vivo* studies are studies that involve people in their own working environment under realistic conditions [20]. Case studies made in an industrial environment are an important type of *in-vivo* study since they allow the analysis of a particular process in the context of a software life cycle [21].

To ensure that the study did not impact the organization's software development process, we restricted ourselves to observing the tasks performed in the team's daily routine and added to their lists the activities required to use the tool, as presented in Figure 7.

Figure 7 presents the activities performed in the organization. The activities in red are the activities we included so that CPM could be used by the team.

At the iteration planning meeting, the team began to include task dependencies. During the execution of the iteration, the team performs daily meetings (daily Scrum meeting). As suggested in Scrum [22], daily meetings should last 15 minutes, where the team discusses what has been done in the last 24 hours, the plan for the next 24 hours, and what task impediments (anything that keeps a team from being productive) occurred. The meeting is held in front of a

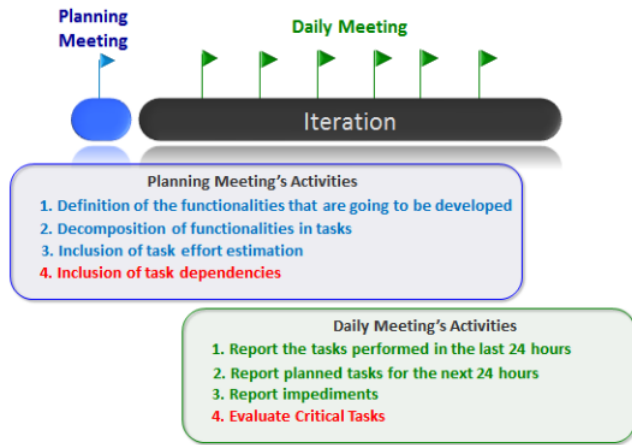


Fig. 7. The activities performed during an iteration.

Kanban board. The visualization of the paths calculated by the AgileCritPath tool was presented at the end of the daily meetings.

During the planning and execution of the iteration, the participants had access to the AgileCritPath tool, and in the daily meetings, the tasks that were part of the critical path were presented to the team. The iteration lasted two weeks, and at the end, the participants were asked to answer the questions directed towards the evaluation of the tool.

At the end of the iteration, the participants completed the Participation Consent and Clearance Form, Participants Characterization Form and the TAM Model Evaluation. The forms were completed individually and without any contact between participants.

C. Discussion

A team of six employees participated in the study. Participation was free and voluntary. The questionnaire was sent to all developers of the company. The participants, in general, have a bachelor's degree, except for one participant, who is in the process of concluding a bachelor's degree. The academic experience varies greatly among the research group since the team has 2 senior, 1 junior and 1 trainee members. All participants are familiar with the basics of project management and are knowledgeable about the Scrum methodology.

From the results of the study, based on the Perceived Utility evaluation criterion, we verified that the participants agreed that the use of the tool could improve the productivity and quality of the work of the team members. All participants recognized the tool as useful for performing their tasks. Regarding the Perceived Usability Facility variable, the team indicated that identifying dependencies between tasks and identifying the Critical Path as reasonably difficult activities. Even though the team has indicated difficulties in identifying dependencies and evaluating the critical path, 75% of respondents felt inclined to use the tool during the planning and execution of tasks. Despite having to make a more significant effort in task dependency identification, the team recognized the benefits that the use of the tool can provide.

The participants were asked to identify the benefits and limitations of the tool. The key highlighted benefits were: the visualization of dependencies between tasks, as a facilitator to determine which tasks should be prioritized, and the improved visualization of the total planned effort to complete critical path tasks. As a tool limitation, the participants recognized that it would be nice if the tool could display estimated versus realized information of the tasks in progress and could provide more metrics for monitoring work progress.

During the daily meetings using the tool, we were able to capture observations made by the team as follows:

- The team observed that using the tool allows everyone on the team to get a sense of what tasks are critical;
- The team found it interesting to leave the critical tasks to the most experienced developers on the project, especially when the deadlines for these tasks are tight;
- Visualizing the critical path is also a way of evaluating the most complex points of the project;
- The team observed that the task priority they provide at the time of task planning did not make sense and, in some cases, the team acknowledges that they might have had a greater gain prioritizing critical path tasks;
- The team evaluated the critical path analysis as a tool complementary to the Kanban framework because in the Kanban framework they could not keep track of the dependencies between the tasks.

The lack of knowledge about dependencies between planned tasks in agile environments emerges in a misaligned business plan that the team may not be able to execute. Also, the lack of awareness about the dependencies that exist between the tasks of the software development process constitutes a possible explanation for inefficient team coordination and project delays.

The AgileCritPath tool enables organizations to use dependency information across tasks to improve task prioritization in agile environments by identifying which dependencies can compromise or block the flow of task execution.

In this study, the evaluation of the usability of the AgileCritPath tool in the industry was conducted. Usability is one of the aspects related to the quality of use of systems, being one of the most important acceptance criteria for interactive applications in general, and in particular for Web applications [23]. The acceptance of technology is related to the quality and use of its systems, the quality of the provided information and user satisfaction [24]. Through the use of the acceptance model, it was possible to evaluate the use of the AgileCritPath tool in an industrial environment.

VI. CONCLUSION

Agile approaches are based on the idea that developers can self-organize and perform their work collaboratively [4]. However, some studies suggest that large and complex software development projects can benefit from the combination of the flexibility inherent in agile teamwork and models that support a plan-oriented structure [25], [26], [27].

Bick et al. [4] and Badampudi et al. [18] propose that organizations should continue to adopt agile methods, including the established practices of traditional methodologies that guarantee more predictability, reliability, stability and effective use of resources. In addition, through analysis of dependencies, the team can evaluate the shorter time for product delivery. Not considering the dependencies between tasks can generate delivery plans that are not aligned with the reality of the organization.

Considering dependencies on software projects is crucial to identifying critical tasks. Critical tasks are tasks that can delay the execution of others and thereby delay the delivery of the product. In this paper, we present an exploratory study that demonstrated that in a real software development scenario, the number of dependencies could be large and difficult to manage. Through the results of the exploratory study, we proposed the use of the Critical Path Method (CPM) concepts to identify critical tasks in agile projects. Critical Path Method has already consolidated in traditional project management models. In Agile Environments, this technique can help the team identify critical tasks, and thus direct their efforts toward tasks that can impact the development of other tasks.

The proposal presented in this article was materialized in the AgileCritPath tool, which allows development teams to have a view of dependencies between the tasks and, therefore, identify at any time which tasks are critical. The tool is compatible with any agile methodology and can support development teams to make decisions easily and quickly. The tool was developed in open source format and is available on GitHub (<https://github.com/RachelVital/CritPath>).

The evaluation of the use of the tool was based on the TAM technological acceptance model. During the evaluation, we were also able to gather insights and encourage the adoption of the tool in the industry. Through the responses obtained in the TAM application, we were able to evaluate that, despite the additional effort to identify the dependencies, the use of the CPM in agile environments proved to be relatively simple and viable in the selected organization. At the end of the evaluation of the use of technology, we could verify that the whole team was able to perceive the utility of the identification of critical tasks.

As future work, we plan to evaluate the use of AgileCritPath in continuous software development and DevOps environments. The application of the acceptance model could also be conducted in other software development companies.

ACKNOWLEDGMENT

The authors thank the Natural Sciences and Engineering Research Council of Canada (NSERC), the Emerging Leaders in the Americas Program (ELAP) and MITACS.

REFERENCES

- [1] J. Lynch. Standish group 2015 chaos report - q&a with jennifer lynch. [Online]. Available: <https://www.infoq.com/articles/standish-chaos-2015>
- [2] M. Cohn, *Agile Estimating and Planning*. Pearson Education, google-Books-ID: BuFWHfRJssC.
- [3] C. D. W. Lomas, J. Wilkinson, P. G. Maropoulos, and P. C. Matthews, "Measuring design process agility for the single company product development process," vol. 9, no. 2, pp. 105–112.
- [4] S. Bick, K. Spohrer, R. Hoda, A. Scheerer, and A. Heinzl, "Coordination challenges in large-scale software development: A case study of planning misalignment in hybrid settings," vol. 44, no. 10, pp. 932–950.
- [5] D. E. Strode, "A dependency taxonomy for agile software development projects," vol. 18, no. 1, pp. 23–46. [Online]. Available: <http://dx.doi.org/10.1007/s10796-015-9574-1>
- [6] M. Korkala and F. Maurer, "Waste identification as the means for improving communication in globally distributed agile software development," vol. 95, pp. 122–140.
- [7] M. Shen, G.-H. Tzeng, and D.-R. Liu, "Multi-criteria task assignment in workflow management systems," in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*. IEEE, pp. 9–pp.
- [8] J. Duggan, J. Byrne, and G. J. Lyons, "A task allocation optimizer for software construction," vol. 21, no. 3, pp. 76–82.
- [9] Y. Jiang and J. Jiang, "Contextual resource negotiation-based task allocation and load balancing in complex software systems," no. 5, pp. 641–653.
- [10] J. Sutherland and K. Schwaber, "The scrum guide. the definitive guide to scrum: The rules of the game."
- [11] W. Aslam and F. Ijaz, "A quantitative framework for task allocation in distributed agile software development," vol. 6, pp. 15 380–15 390.
- [12] H. Alaidaros, M. Omar, and R. Romli, "Identification of criteria affecting software project monitoring task of agile kanban method," in *AIP Conference Proceedings*, vol. 2016. AIP Publishing, p. 020021.
- [13] Y. Lee, K. A. Kozar, and K. R. Larsen, "The technology acceptance model: Past, present, and future," vol. 12, no. 1, p. 50.
- [14] VersionOne. 12th annual state of agile report. [Online]. Available: <https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report>
- [15] M. L. Despa, "Comparative study on software development methodologies," vol. 5, no. 3, pp. 37–56.
- [16] Hazır, "A review of analytical models, approaches and decision support tools in project monitoring and control," vol. 33, no. 4, pp. 808–815.
- [17] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," vol. 14, no. 2, pp. 131–164. [Online]. Available: <http://link.springer.com.ez29.capes.proxy.ufrj.br/article/10.1007/s10664-008-9102-8>
- [18] D. Badampudi, S. A. Fricker, and A. M. Moreno, "Perspectives on productivity and delays in large-scale agile projects," in *International Conference on Agile Software Development*. Springer, pp. 180–194.
- [19] F. D. Davis, "A technology acceptance model for empirically testing new end-user information systems : theory and results," <http://hdl.handle.net/1721.1/15192>, 7 1986, thesis (Ph. D.).—Massachusetts Institute of Technology, Sloan School of Management, 1986.; MICROFICHE COPY AVAILABLE IN ARCHIVES AND DEWEY.; Bibliography: leaves 233-250.
- [20] G. H. Travassos and M. O. Barros, "Contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering," in *2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering*, pp. 117–130.
- [21] F. Shull, J. Carver, and G. H. Travassos, "An empirical methodology for introducing software processes," in *ACM SIGSOFT Software Engineering Notes*, vol. 26. ACM, pp. 288–296.
- [22] P. Deemer, G. Benefield, C. Larman, and B. Vodde. The scrum primer version 2.0.
- [23] E. Insfran and A. Fernandez, "A systematic review of usability evaluation in web development," in *International Conference on Web Information Systems Engineering*. Springer, pp. 81–91.
- [24] S. Petter, W. DeLone, and E. R. McLean, "The past, present, and future of" IS success"," vol. 13, no. 5, p. 341.
- [25] J. B. Barlow, J. S. Giboney, M. J. Keith, D. W. Wilson, and R. M. Schuetzler. Overview and guidance on agile development in large organizations.
- [26] L. Cao, K. Mohan, P. Xu, and B. Ramesh, "How extreme does extreme programming have to be? adapting XP practices to large-scale projects," in *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*. IEEE, pp. 10–pp.
- [27] N. Ramasubbu, A. Bharadwaj, and G. K. Tayi, "Software process diversity: conceptualization, measurement, and analysis of impact on project performance."

Evaluating Software Developers' Acceptance of a Tool for Supporting Agile Non-Functional Requirement Elicitation

Felipe Ramos[§], Antônio Pedro[¶], Marcos Cesar[¶], Alexandre Costa[§],
Mirko Perkusich[¶], Hyggo Almeida[‡] and Angelo Perkusich^{‡‡}

Intelligent Software Engineering Group, Federal University of Campina Grande,
Campina Grande - PB, Brazil, Zip Code 58429-140

[§]CAPES Foundation, Ministry of Education of Brazil, Brasilia - DF, Zip Code 70.040-020

[§]{feliperamos, antonioalexandre}@copin.ufcg.edu.br,

[¶]{antonio.abreu, marcos.cesar, mirko.perkusich}@embedded.ufcg.edu.br

[‡]hyggo@dsc.ufcg.edu.br, ^{‡‡}perkusic@dee.ufcg.edu.br

Abstract—Due to the need for flexibility to requirements changes, agile software development methods have been attracting the attention of academic and industrial domains. Unlike traditional approaches, agile methods focus on the rapid delivery of business value to customers through empirical and incremental development processes. Despite being effective in delivering quality functional requirements, agile practices generally neglect non-functional requirements until the later stages of software development. However, neglecting non-functional requirements during requirements analysis can lead to project failures. In this paper, we present the NFRec tool, which aims to support software developers in the elicitation of non-functional requirements in the context of agile software development. Additionally, we report the results from a case study to evaluate the acceptance of the NFRec tool from the point of view of software developers of four projects from a Brazilian software company. To gather information about the tool acceptance, we applied a questionnaire based on the indicators from the Technology Acceptance Model. Overall, the four teams considered the NFRec tool useful and easy to use for supporting the management of non-functional requirements in agile projects.

Keywords—Non-functional requirements; agile requirement engineering; supporting tool; empirical study; technology acceptance.

I. INTRODUCTION

Since the declaration of the Agile Manifesto in 2001, academic [8] and industrial [16] communities have devoted considerable attention to agile software development (ASD) methods, such as Scrum and eXtreme Programming. One of the key aspects of the ASD is to rapidly adapt to volatile requirements [8], following a development process that places a great emphasis on frequent delivery of business value for customers [8].

Although agile practices are considered effective in the delivering of valuable functional requirements (FRs) [14], non-functional requirements (NFRs) are commonly neglected until the later stages of software development [14]. However, neglecting NFRs can lead to software failure [2], since NFRs are often more critical to determine the perceived success or failure of a software product than FRs [9].

By addressing this problem, some studies proposed techniques to support the NFR elicitation in the agile context [10], [11], [12], [13]. Maiti and Mitropoulos [12], [13] presented a methodology to collect NFRs metadata from software requirements artifacts such as documents and images available in the initial stages of projects. Farid et al. [10], [11] proposed a solution based on decision trees to predict NFRs from future iterations of agile projects based on the metadata collected with the methodology proposed in Maiti and Mitropoulos [12], [13]. However, to the best of our knowledge, none of these works proposed the use of historical data of previous projects to provide suggestions of NFRs for ongoing ones. Additionally, there is a lack of empirical evaluations of proposed solutions in real industrial environments.

In our previous work [15], we proposed a non-functional requirement recommendation system (RS) for Scrum-based projects, which is based on the analysis of information from past projects. As a result, our solution achieved a recall rate of up to 81%, showing the feasibility of automating the definition of NFRs through historical data of Scrum-based projects. To allow the applicability of the proposed recommendation system on industrial environments, we developed a tool based on an improved version of the solution presented in [15], called NFRec.

In the current paper, we focus on the empirical assessment of the NFRec tool and report the results obtained through a case study to evaluate its acceptance in a real industrial environment. For matters of validation, our work focus on Scrum, which is the most popular agile method [17]. As a result, the subjects of the case study considered the NFRec tool useful and easy to use for supporting the management of non-functional requirements in agile projects.

This paper is organized as follows. Section II presents a background and a motivation to use a supporting tool for NFRs elicitation in ASD context. Section III shows the NFRec tool. Section IV presents details about the empirical evaluation. Section V discusses the threats to validity. Finally, Section VI details our conclusions and perspectives for future works.

II. BACKGROUND AND MOTIVATION

Non-functional requirements play a key role in the successful development of software products [2]. Moreover, they are considered as a factor of differentiation among software products that present similar FRs [1].

In this sense, Bourimi et al. [3] proposed a framework that aims to conceptually impose the early consideration of NFRs, by considering the adoption of the stakeholder of NFRs in the Scrum team. In contrast, Farid [7] proposed a framework for modeling NFRs in the context of ASD.

Although the presented techniques provide a means for identifying NFRs on agile projects, there is still room for improvement. For example, solutions such as the ones proposed by Bourimi et al. [3] and Farid [7] present only a conceptual reinforcement or/and the addition of artifacts and roles in the agile process. Therefore, no solutions are proposed to support automated NFR elicitation. In contrast, solutions presented by Farid et al. [10], [11] and Maiti and Mitropoulos [12], [13] require that project documents and images are available on early stages of the software development, which is not common when dealing with ASD. The presented needs have been considered in the NFRec tool.

III. NFRREC TOOL

In this section, we present the NFRec tool, which aims to support developers in the early elicitation of NFRs during the Sprint Planning Meetings¹. NFRec comprises the following two activities: (i) Structuring of Project Information; and (ii) NFR Recommendation.

A. Structuring of Project Information

To enable the generation of the NFRs recommendations, it is necessary to structure the projects' information to guarantee the retrieval of information by the recommendation system. In the NFRec tool, user stories and project profiles are structured based on the assignment of categories and tags.

The first activity accomplished in the NFRec tool by Scrum teams is to create project profiles based on the assignment of tags referring to the five factors that directly affect the definition of NFRs [15], i.e., platform (e.g., web, mobile, embedded, desktop, etc.), project domain (e.g., health, banking, etc.), project objective (e.g., product or prototype), software architecture (e.g., client-server, MVC, multilayered, etc.), and technology tags, which represent basic technologies for the development of a software product (e.g., programming language, database, etc.). In Figure 1, we present an example of a structured project profile created in the NFRec tool. The project is from the domain *Development Tools* (1), its objective is to develop a *Product* (2), the software architecture is *Client-server* (3), the platform is *Web* (4) and the technologies used are the programming languages *Java* and *JavaScript* (5) and the frameworks *Angular* and *Spring Boot* (6). We highlight that previously stored tags must be reused by Scrum teams to avoid data inconsistencies.

Besides project profiles structuring, all user stories (USs) specified in the NFRec tool must be categorized to enable the retrieval of information by the RS. Therefore, during the Sprint

¹Sprint Planning Meeting is an event of the Scrum in which the Scrum team plans the work to be performed in the Sprint.

Fig. 1: Example of a structured project profile created in the NFRec tool

Planning Meetings, developers use the NFRec tool to create and store USs, which are classified by a category (module) and a subcategory (operation) that indicate the purpose of the FR specified by the US. In Figure 2, we present an example of a structured user story created in the NFRec tool. The US is classified with the module *Registration* (1) and the operation *Retrieve data* (2). We highlight that previously stored categories (i.e., modules and operations) must be reused by development teams to avoid data inconsistencies.

B. NFR Recommendation

For each US defined by the development team, the NFRec tool recommends a list of NFRs based on historical data

New Issue

Title *
US04: As a Product Owner I wish to View test case recommendation so that I can obt

Module * ①
Registration x

Operation * ②
Retrieve data x

As a
Product Owner x

I wish
View test case recommendation

So that
I can obtain test cases suggestion based on functional requirements

Type *
User Story

Estimated Points *
0

Status *
Todo

Scope *
Planned

Start date
Start date

Due date
Due date

Assigned To
None Selected

Delivered on Build
[Dropdown]

Requirement *
Test cases recommendation

Acceptance Criteria
Enter Acceptance Criteria

Next

Fig. 2: Example of a structured user story created in the NFRec tool

analysis. NFRs are represented in the tool by a type, an attribute, and a sentence. The classification with type and attribute follows the indications presented by Mairiza et al. [9]. Additionally, NFR sentences are written following the indications presented by Eckhardt et al. [5], [6].

In Figure 3, we present an example of NFR recommendations presented in the NFRec tool, in which three NFRs are suggested for a US of the module *Registration* (4) and the operation *Retrieve data* (5). In the example, the developers have already accepted/considered one of the three recommendations

Non-functional requirements

Module: Registration ④
Operation: Retrieve data ⑤

type	attribute	sentence
security ①	access control ②	the system must provide the capability for users see just the information they have permission to access. ③
reliability	availability	the system must notify users and customers when the function they are accessing is unavailable.
performance	response time	in 100% of all cases, the system must have a mean response time of <= 3s between event <search items> and event <display text information of listed items>. it is assumed that there are 200 concurrent users.

Back **Finish**

Fig. 3: Example of a NFR recommendation presented in the NFRec tool

(6), a NFR of type *security* (1), attribute *access control* (2), and sentence “*The system must provide the capability for users to see just the information they have permission to access.*” (3).

By using the tool, developers can visualize suggestions of NFRs for each US of current or future Sprints. Thus, NFRs can be considered early in the software development process, mitigating the risk of negligence with non-functional requirements resulted from agile practices.

IV. EMPIRICAL EVALUATION

To evaluate the practical use of the NFRec tool, we conducted a case study in four ongoing Scrum-based projects from a Brazilian software company. We intend to: (i) assess the acceptance of the NFRec tool by agile software developers; (ii) evaluate the precision of generated recommendations.

A. Case Study Design

We performed an embedded case study [18] in which each project was considered a unit of analysis and each development team the subject of study. This step of the research lasted a month and each project executed two Sprints of 15 days during this period. Meanwhile, we collect information from different recommendation scenarios, and hence, we consider a sufficient period to answer the research questions.

The overall objective of the case study is to evaluate the cost-benefit of the NFRec tool from the perspective of development teams, regarding the support of non-functional requirements management. To accomplish that, we formulate the following research questions (RQs):

- **RQ₁**: is the NFRec tool useful to assist in eliciting non-functional requirements in Scrum-based software projects?
- **RQ₂**: is the cost to use the NFRec tool in Scrum-based software projects acceptable?
- **RQ₃**: what is the precision of the recommendations of the NFRec tool?

Therefore, we consider the cost-benefit of the NFRec tool worthwhile if the questions are positively answered.

As previously stated, we consider four software projects as study analysis units, referred as Project A, Project B, Project C and Project D. All of them consisting of Web information systems with the following scopes:

- **Project A:** development of a system with a cloud service and a Web client to enable independent or shared writing of poetry. It is composed of two developers;
- **Project B:** development of a Web client for generating graphics resources such as badges and business cards. It is composed of three developers;
- **Project C:** development of a system with a cloud service and a Web client to support the management of training projects through the management of students' activities and schedules, selections of participants to projects, etc. It is composed of five developers;
- **Project D:** development of a system with a cloud service and a Web client to assist the building and executing of Bayesian Networks. It is composed of five developers.

Prior to the case study, all projects performed requirements management during the Sprint Planning Meetings using a tool without NFR recommendation. Therefore, they did not perform any direct activity to define NFRs through the tool. They described them as acceptance criteria, DoD items, functional requirements, failures identified by test cases, etc.

To gather data for evaluating the acceptance of the NFRec tool, we apply a questionnaire based on the Technology Acceptance Model [4]. TAM aims to explain why individuals choose to adopt or not adopt a specific technology when accomplishing a task and it is based on two variables: Perceived Usefulness (PU) and Perceived Ease of Use (PEU). PU is related to the degree to which an individual believes that the use of a certain technology would increase his/her performance in the work. In contrast, PEU refers to the degree to which an individual believes that the use of a certain technology would be free of mental and physical effort.

Therefore, by adapting the TAM to the context of this work, we formulate 14 questions to evaluate the PU of the NFRec tool (RQ_1) and 14 to assess its PEU (RQ_2). For each question, we use a five-level Likert scale to collect participants' responses. The scale adopted the values: (1) Strongly Disagree, (2) Disagree, (3) Neither agree nor disagree, (4) Agree, and (5) Strongly Agree. In addition, we calculate the precision of the recommendations processed during the case study (RQ_3).

Overall, we collected 4 answers for the questionnaire, i.e., one for each development team. On average, the teams of projects B and C have one year of experience in Scrum-based software projects. On the other hand, the teams of projects A and D had a mean of three years of experience in the same type of project.

B. Case Study Execution

To run the case study, we make the NFRec tool available to the teams of each project via a web link. The execution of the study comprised two stages: (i) training, and (ii) execution.

During the training phase, we presented concepts about the structuring of project information and the non-functional requirements recommendation, followed by a demonstration of the NFRec tool. The training lasted 1 hour. After this phase, the teams started using the NFRec tool at their projects' Sprint Planning Meetings, i.e., a real evaluation scenario in the industry.

First, they used the tool to create the profiles of their respective projects based on the assignment of tags referring to the five factors that affect the definition of NFRs (i.e., application domain, platform, project objective, software architecture, and technologies [15]). None of the teams had reported any troubles at this step.

Next, for each current Sprint, the teams used the NFRec tool to support the requirement management process during the Sprint Planning Meetings, in which they created structured USs. For each created US, the teams received NFRs recommendations and they could freely interact with the tool, accepting or rejecting suggestions. The duration of the meetings did not change in any of the projects, i.e., they continued within the planned interval of 1 hour. In the following, we present the activities carried out in the first moment of the case study.

In the first observed Sprint of Project A, the team selected one US, which received two NFR recommendations. The team accepted only one of them. Therefore, the RS achieved a precision rate of 50% for the corresponding Sprint from Project A.

For the first observed Sprint of Project B, the team selected two USs. For the first described US, the NFRec tool generated six NFR recommendations. The team accepted five of them. In contrast, for the second US, the tool generated two NFR recommendations, but the team accepted only one of them. Therefore, the RS achieved a precision rate of 75% for the first evaluated Sprint from Project B.

In the first observed Sprint of Project C, the team selected three USs and the NFRec tool generated three NFR recommendations for each one them. For two of the USs, the team accepted all the suggestions. On the other hand, for the remaining one, the team accepted only two of the recommendations. Therefore, the RS achieved a precision rate of 88.88% for the respective Sprint from Project C.

Finally, in the first observed Sprint of Project D, the team selected three USs. Two of them received three NFR recommendations, which were accepted by the team. In contrast, for the remaining one, the tool presented two suggestions of NFRs, of which only one was accepted by the team. Therefore, the RS achieved a precision rate of 87.5% in the first evaluated Sprint from Project D.

We highlight that the Product Owners of each project validated the recommendations accepted by the development teams. However, we did not evaluate whether the constraints specified by the recommended NFRs were considered in the development of the USs within the current Sprints since this issue is not part of the scope of this work. In the following, we present the activities carried out in the second moment of the case study.

In the second observed Sprint of Project A, the team selected two USs during the Sprint Planning Meeting. The NFRec tool presented one recommendation for one of the US

and three for the other. The team accepted all the suggestions. Therefore, the RS achieved a precision rate of 100% in the second evaluated Sprint from Project A.

In the second observed Sprint of Project B, the team selected two USs during the Sprint Planning Meeting. The NFRec tool recommended one NFR for two of them. For the other one, the tool presented two recommendations. Overall, the team accepted two of the four suggestions. Therefore, the RS achieved a precision rate of 50% in the second first Sprint evaluated from Project B.

In the second observed Sprint of Project C, the team selected two USs. The NFRec tool recommended one NFR for the first described US and two for the later. The team accepted all the suggestions. Therefore, the RS achieved a precision rate of 100% for the corresponding Sprint.

Finally, for the second observed Sprint of project D, the development team selected three USs and the NFRec tool generated two NFR recommendations for each one of them. The team accepted all the suggestions for two of the USs but rejected one of the recommendations for the remaining one. Therefore, the RS achieved a precision rate of 83.33% for the corresponding Sprint from Project D.

At the end of the case study, all subjects filled in the questionnaire regarding their acceptance of the NFRec tool.

C. Results and Discussion

After the case study execution, we calculated the precision of the recommendations generated during the observed period. In Table I, we present the results of the overall precision and the precision for each project. Overall, the NFRec tool recommended 44 NFRs, of which 36 were accepted by the developers, resulting in a precision rate of 81.8% of the recommendations for the 20 considered USs. Therefore, we considered the results promising, concluding that RQ_3 was positively answered. Among the four evaluated projects, we observed a higher precision rate in Project C (91.7 %), which is composed of professionals with previous experience in the use of the structured model of USs. On the other hand, we observed the lowest precision rate in Project B (66.7 %), which is the only one of the four projects that it is not developing a cloud service (*back-end*). Therefore, some recommendations were rejected for suggesting back-end assumptions/constraints such as validating the integrity of data sent to the server. Additionally, the team of Project B reported difficulties during the classification of the USs with modules and operations.

TABLE I: Results of precision calculated from the data obtained in the case study

Project	Num. USs	Num. accep. NFRs	Num. rec. NFRs	Precis.
Project A	3	5	6	83,3%
Project B	5	8	12	66,7%
Project C	5	11	12	91,7%
Project D	7	12	14	85,7%
Total	20	36	44	81,8%

As mentioned before, at the end of the case study, each development team answered a questionnaire to assess the acceptance of the solution regarding the perceived usefulness and perceived ease of use. To evaluate the responses, we summarized them as follows: Likert scale values represented by (1) and (2) were considered as indicative of disagreement

(*Disagreement*); (3) as indicative of neutrality (*Neutral*); and (4) and (5) as indicative of agreement (*Agreement*).

In Tables II and III, we present the data collected for PU and PFU, respectively. Positive responses to variables are highlighted in bold. The maximum number of answers per question is four since we considered four subjects in the study. Therefore, as we formulate 14 questions for each TAM variable, the maximum number of responses per variable is 56. The final result for each analyzed variable is given by the sum of the answers per positive indicative (i.e., *Agreement*) divided by the maximum number of responses to the corresponding variable (i.e., 56).

By analyzing the data of the Table II, we verify that the research participants showed good acceptance for 13 of the 14 analyzed items of PU. Only items PU9 and PU12 did not receive mostly positive evaluations. For PU9, participants gave three neutral answers and one positive. In contrast, for PU12, they gave two positive responses and two neutral ones. Overall, the PU was positively assessed in 48 of 56 possible responses. The evaluation demonstrated an acceptance of the perceived usefulness of 85.7%. Therefore, it is possible to state that the research participants considered the tool useful (RQ_1).

By analyzing the data of the Table III, we verify that the respondents showed good acceptance for 12 of the 14 analyzed items of PFU. Only the items PFU1, PFU11, PFU12, PFU13, and PFU14 did not receive mostly positive evaluations. For PFU1 and PFU13, the respondents gave two positives and two negative responses. In contrast, for PFU11, PFU12, and PFU14, they gave two positives and two neutral responses. Overall, the respondents positively assessed PFU in 41 of 56 answers, i.e., an acceptance of the perceived ease of use of 73.2%. Initially, some developers encountered difficulties in structuring USs and defining modules and operations. This fact may explain why PFU acceptance values were lower than those of PU. Even so, it is possible to state that the research participants considered the tool easy to use (RQ_2).

TABLE II: Results for perceived usefulness from TAM

ID	Item	Agreement	Neutral	Disagreement
PU1	Job Difficult Without	4	0	0
PU2	Control Over Work	4	0	0
PU3	Job Performance	4	0	0
PU4	Addresses My Needs	4	0	0
PU5	Saves Me Time	3	1	0
PU6	Work More Quickly	4	0	0
PU7	Critical to My Job	4	0	0
PU8	Accomplish More Work	3	1	0
PU9	Cut Unproductive Time	1	3	0
PU10	Effectiveness	4	0	0
PU11	Quality of Work	3	1	0
PU12	Increase Productivity	2	2	0
PU13	Makes Job Easier	4	0	0
PU14	Useful	4	0	0

V. THREATS TO VALIDITY

In this work, we consider the classification of validity threats proposed by Wohlin et al. in [18]. In what follows, we present the identified threats to validity.

Conclusion validity threats. We conducted the case study for two 15-day Sprints only, which represents a conclusion validity threats since we could get different answers with a longer period. However, to mitigate this threat, we evaluated four teams simultaneously, which returned similar results. In

TABLE III: Results for perceived ease of use from TAM

ID	Item	Agreement	Neutral	Disagreement
PFU1	Confusing	2	0	2
PFU2	Error Prone	0	1	3
PFU3	Frustrating	0	0	4
PFU4	Dependence on Manual	0	1	3
PFU5	Mental Effort	0	0	4
PFU6	Error Recovery	4	0	0
PFU7	Rigid and Inflexible	1	1	2
PFU8	Controllable	4	0	0
PFU9	Unexpected Behavior	0	0	4
PFU10	Cumbersome	0	1	3
PFU11	Understandable	2	2	0
PFU12	Ease of Remembering	2	2	0
PFU13	Provides Guidance	2	0	2
PFU14	Easy to Use	2	2	0

addition, we collected data regarding the different types of USs, i.e., we could observe different recommendation scenarios during the case study.

Internal validity threats. On average, developers from two of the four projects had just one year of experience. This fact can lead to a threat to internal validity, since they may not be mature enough to properly answer the questionnaire. To mitigate this threat, developers of each project answered the questionnaire together, and hence, they had the opportunity to discuss their responses with each other, aggregating their knowledge. Another threat to internal validity refers to the construction of the questionnaire. However, to mitigate this problem, we generated it based on the TAM, which is a validated and extensively used model in the literature to evaluate new technologies.

Construct validity threats. Factors related to the type of project (e.g, scope, complexity, familiarity with technology, team experience, etc.) might affect the acceptability of the tool, which can lead to a threat to construct validity. To mitigate this problem, we tried to select projects with different scopes and domains. However, we reinforce that further research is needed to investigate this threat to validity.

External validity threats. The case study was carried out with four projects from the same company and, consequently, it is not possible to generalize the obtained results to other companies that use Scrum. However, as future work, we intend to continue the empirical evaluation of the NFRec tool in more projects from different companies.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented the NFRec tool for supporting agile non-functional requirement elicitation. We reported the results of a case study to evaluate the acceptance of the tool from the point of view of development teams of four software projects from a Brazilian software company.

The empirical evaluation indicated that the NFRec tool seems to be able to accurately recommend NFRs, responding positively to the research question RQ_3 . Furthermore, the results of the case study obtained through the questionnaire showed that most of the subjects considered the NFRec tool useful and easy to use for supporting the elicitation of non-functional requirements, responding positively to the research questions RQ_1 and RQ_2 .

For future work, we intend to replicate this study with a greater number of subjects, analyzing different projects from distinct software companies to mitigate the validity threats.

ACKNOWLEDGMENT

The authors would like to thank CAPES for supporting this work.

REFERENCES

- [1] B. M. Aljallabi and A. Mansour. Enhancement approach for non-functional requirements analysis in agile environment. In *2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICNEEE)*, pages 428–433, Sept 2015.
- [2] V. Bajpai and R. P. Gorthi. On non-functional requirements: A survey. In *Electrical, Electronics and Computer Science (SCEECS), 2012 IEEE Students' Conference on*, pages 1–4, March 2012.
- [3] M. Bourimi, T. Barth, J. M. Haake, B. Ueberschär, and D. Kesdogan. AFFINE for enforcing earlier consideration of NFRs and human factors when building socio-technical systems following agile methodologies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6409 LNCS:182–189, 2010.
- [4] F. D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, pages 319–340, 1989.
- [5] J. Eckhardt, A. Vogelsang, and H. Femmer. An approach for creating sentence patterns for quality requirements. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 308–315, Sep. 2016.
- [6] J. Eckhardt, A. Vogelsang, H. Femmer, and P. Mager. Challenging incompleteness of performance requirements by sentence patterns. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 46–55, Sep. 2016.
- [7] W. M. Farid. The normap methodology: Lightweight engineering of non-functional requirements for agile processes. In *Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference - Volume 01, APSEC '12*, pages 322–325, Washington, DC, USA, 2012. IEEE Computer Society.
- [8] R. Hoda, N. Salleh, J. Grundy, and H. M. Tee. Systematic literature reviews in agile software development: A tertiary study. *Information and Software Technology*, 85:60 – 70, 2017.
- [9] D. Mairiza, D. Zowghi, and N. Nurmiliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing, SAC '10*, pages 311–317, New York, NY, USA, 2010. ACM.
- [10] R. R. Maiti, A. Krasnov, and D. M. Wilborne. Agile software engineering & the future of non-functional requirements. *Journal of Software Engineering Practice*, 2(1):1–8, december 2018.
- [11] R. R. Maiti, A. Krasnov, and M. Wilborne. Predicting nfrs in agile software engineering. In *Proceedings of the 19th Annual SIG Conference on Information Technology Education, SIGITE '18*, pages 161–161, New York, NY, USA, october 2018. ACM.
- [12] R. R. Maiti and F. J. Mitropoulos. Capturing, eliciting, predicting and prioritizing (cepp) non-functional requirements metadata during the early stages of agile software development. In *SoutheastCon 2015*, pages 1–8, April 2015.
- [13] R. R. Maiti and F. J. Mitropoulos. Capturing, eliciting, and prioritizing (cep) nfrs in agile software engineering. In *SoutheastCon 2017*, pages 1–7, March 2017.
- [14] B. Ramesh, L. Cao, and R. Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010.
- [15] F. Ramos, A. Costa, M. Perkusich, H. Almeida, and A. Perkusich. A non-functional requirements recommendation system for scrum-based projects. In *The 30th International Conference on Software Engineering and Knowledge Engineering*, 2018.
- [16] S. Stavru. A critical examination of recent industrial surveys on agile method usage. *Journal of Systems and Software*, 94:87 – 97, 2014.
- [17] VersionOne. *12th Annual State of Agile Survey*, 2018. Acessado em: 19 de dezembro de 2018. <https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report>.
- [18] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

Automatic Generation of Virtual Assistants from Databases using Active Ontologies

Martin Blersch and Sebastian Weigelt and Walter F. Tichy

Institute for Program Structures and Data Organization

Karlsruhe Institute of Technology

Karlsruhe, Germany

blersch@kit.edu, weigelt@kit.edu, tichy@kit.edu

Kevin Angele

Semantic Technology Institute

University of Innsbruck, Innsbruck, Austria

and *ONLIM GmbH, Telfs, Austria*

kevin.angele@sti2.at

Abstract—Virtual assistants such as Siri or Google Assistant are omnipresent. However, their development remains costly. One must either manually model the problem domain or provide thousands of labeled samples.

We propose to automatically create virtual assistants based on Active Ontologies for interacting with databases. Our approach generates Active Ontologies; we use the database structure to derive a concept hierarchy and database values together with synonyms to extract information from user queries. Our approach also learns common phrases from samples, e.g. from existing Dialogflow agents. We extract pre- and postfixes and attach them to concepts, e.g. *at* to detect a succeeding location. The generated Active Ontologies reply to previously unseen and composed requests. The approach is not limited to virtual assistants but can be applied to any system with a textual or voice-based conversational interface such as chatbots.

We evaluate our approach in three domains: tourism, hotel, and web cams. The study shows that automatically generated Active Ontologies extract relevant information from user utterances with a precision of 58%. The precision increases to 79% (recall 46%, F_1 58%) when we use sample utterances. Our approach successfully transfers between domains, e.g. we learn phrases from the tourism domain and use them to reply to hotel requests without any adjustments.

I. INTRODUCTION

Virtual assistants and other systems with conversational interfaces (CI) are used by an ever-growing number of people. Users ask Siri for their next meeting, talk to chatbots, or tell Alexa to turn on the lights. While these systems are a blessing for casual users, they remain a curse for developers. They are complicated to build and hard to maintain. Today’s virtual assistants are either trained on thousands or even millions of (manually) labeled samples, or their linguistic competence is modeled manually, i.e. they are built by domain experts. Both require serious manual effort to make the system appear human-like to the user. However, users will soon expect CIs for all kinds of applications. Thus, the efficient creation of CIs (i.e. developer assistance, transferability, automation, etc.) will become one of the major challenges in software engineering.

We propose to generate systems with CIs largely automatically. As underlying technology we use Active Ontologies [1], [2] (AO) that originally were at the core of Apple’s Siri. AOs are hierarchical domain models equipped with processing

rules. In terms of structure they are trees, where the leaves react to words in user utterances. The inner nodes join information and the root creates a services call, e.g. a restaurant reservation.

As a prerequisite for our generation process, we assume that a service provider stores information in a database, e.g. the addresses, ratings, and names of restaurants. Having this information, our AOs are supposed to answer requests such as, “Show me restaurants in Lisbon.” We automatically generate AOs from the databases in two steps. First, we use the structure of the database to infer the concept hierarchy of the AO. Second, we add leaf nodes to provide the AO with linguistic competence. We generate those from database values and add synonyms obtained from Wiktionary¹. Optionally, if a data set of sample user utterances is present, we can further improve the linguistic competence. We extract related phrases for each concept. For example, we can learn the phrase *where can I find* for the concept `location` from examples like, “Where can I find an Italian restaurant.” Our generated AOs reply to previously unseen request, e.g. finding a hotel in a city (which was learned from *finding* web cams and listing tourist attractions *in a city*). Moreover, they correctly respond to complex requests composed of two or more simple requests, e.g. asking for both, the location and opening hours of a restaurant, at the same time. The developer reviews the result of the generation process and adapts the AO. Usually, this involves altering the type of inner nodes and adding common phrases to leaf nodes.

The remainder is structured as follows. First we introduce the basic elements of Active Ontologies in section II. Then, we discuss related work in section III. In section IV we present our approach and the generative process in detail together with a discussion about its inherent limitations. Afterwards, we evaluate our approach in section V. We conclude our work and discuss further improvements in section VI.

II. ACTIVE ONTOLOGIES

Active Ontologies were first proposed by Guzzoni et al. [1], [2]. Originally, they were used to build virtual assistants. However, they can be used as a generic CI. We present

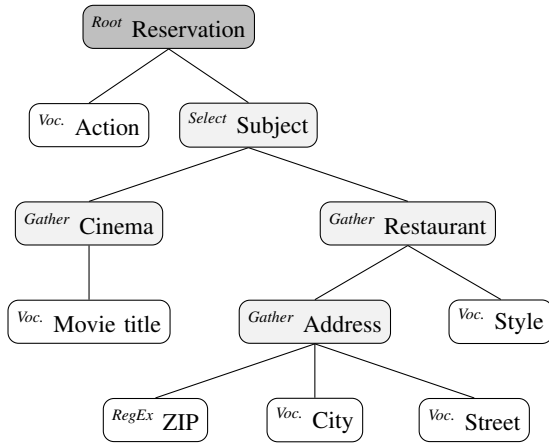


Fig. 1. An AO for the problem domain “reservations”. The AO is can create service calls to cinema and restaurant reservation systems. The root node is colored in gray, the inner nodes in light gray, and leaf nodes are white.

the fundamental elements and basic processing of AOs. All upcoming examples refer to the AO depicted in Figure 1. Detailed descriptions may be found in the above reference.

AOs combine concept hierarchies with processing rules. On the one hand, they model the domain as an ontology. On the other hand, AOs process natural language requests and turn them into service calls. The concepts and relations form trees, i.e. an AO is a concept hierarchy of a problem domain. Information is processed bottom-up. The leaf nodes react to words in the user utterances. The concept nodes (i.e. the inner nodes) join information and gradually create an abstract representation of the utterance. Finally, the root node gathers all information and initiates a service call. Basically, all nodes react to input as follows. When they receive a message (i.e. a piece of information), they decide whether they send information upwards or not. For the creation of new messages, nodes often use parts of the received information. Additionally, they attach a confidence to the fired message. This may support the decision-making processes of nodes at higher levels. The decision whether, what, and with which confidence they fire depends on the particular usage. All of that is implemented by rule sets. In the following we describe common node types.

Basic leaf nodes (called *vocabulary nodes*) simply match a predefined set of keywords, e.g. city names. Often, *pre-* and *postfix nodes* are used. They define a pre-/succeeding word or phrase, e.g. the prefix “from” to match the departure airport for flight booking systems. Another common type of leaf node is the *regex node* that matches input with a regular expression, e.g. to detect ZIP codes. Usually, there are two types of inner nodes: *gather nodes* and *selection nodes*. Gather nodes simply collect information sent by their child nodes; e.g. an *address* gather node might collect *ZIP*, *city*, and *street* facts. Selection nodes decide which information will be passed on; e.g. a *subject* selection node might decide whether the user talks about restaurants or cinemas. Most commonly, selection nodes decide on the basis of the confidences of the respective inputs.

However, other strategies are possible. The most common type of root node is the *call node*. It creates a service call and passes it to the *service broker*. The service broker is a sub-system that selects the most suitable service provider(s) for the call. A language generation module post-processes the response to the service call. The result is presented to the user.

III. RELATED WORK

In this section, we first review proprietary virtual assistants (see subsection III-A). Then, we present platforms for developers to create systems with CIs, e.g. chatbots and the like (see subsection III-B). Finally, we discuss work from the research area natural language interfaces to databases (see subsection III-C).

A. Proprietary Virtual Assistants

Virtual assistants both for home environments (e.g., Amazon Echo, Google Home) and mobile use (e.g., Apple Siri, Google Assistant, Microsoft Cortana) are omnipresent. However, little is known about the technology behind these assistants. US patent no. 8,677,377 [3] suggests that Apple’s Siri makes use of Active Ontologies. Amazon’s Alexa interacts with third-party services through so-called “skills”. Google uses a knowledge graph to answer user queries² and Amazon states that they use “deep learning technologies”³ to develop Alexa.

B. Platforms for Conversational Interfaces

Besides proprietary assistant systems, all major companies provide platforms for developers to create CIs, e.g. IBM Watson⁴, Microsoft LUIS⁵, Facebook’s WIT⁶, Amazon Lex⁷, and Google’s Dialogflow⁸.

Dialogflow provides an API for developers to add natural language processing capabilities to applications. One can build CIs to create chatbots and the like. Developers build so-called *agents*. Agents determine the user’s intent from an utterance. Each agent deals with one or more *intents*, e.g. requests for weather forecasts or web cams. When an agent grasps an intent, it extracts relevant information and passes it to a connected service. Since users may express intents in different ways, it is necessary to provide Dialogflow agents with a variety of different phrases for each intent. The developer does not only have to provide the phrases but also annotate actions and parameters in all phrases. One must also specify the parameter mapping, i.e. which word must be translated to which parameter in the service call.

Almond [4], developed by the Stanford University, is an open and crowd-sourced platform to build virtual assistants. Almond is composed of three modules: a virtual assistant, the knowledge base *Thingpedia*, and the runtime environment

²Andreas Blumauer, Semantic Web Company: <https://semantic-web.com/2018/08/23/knowledge-graphs-connecting-dots-increasingly-complex-world/>

³Amazon Alexa: <https://developer.amazon.com/de/alexa/science/>

⁴Watson Assistant: <https://www.ibm.com/cloud/watson-assistant/>

⁵LUIS: <https://www.luis.ai/>

⁶wit.ai: <https://wit.ai/>

⁷Amazon Lex – Build Conversational Bots: <https://aws.amazon.com/lex/>

⁸Dialogflow: <https://dialogflow.com/>

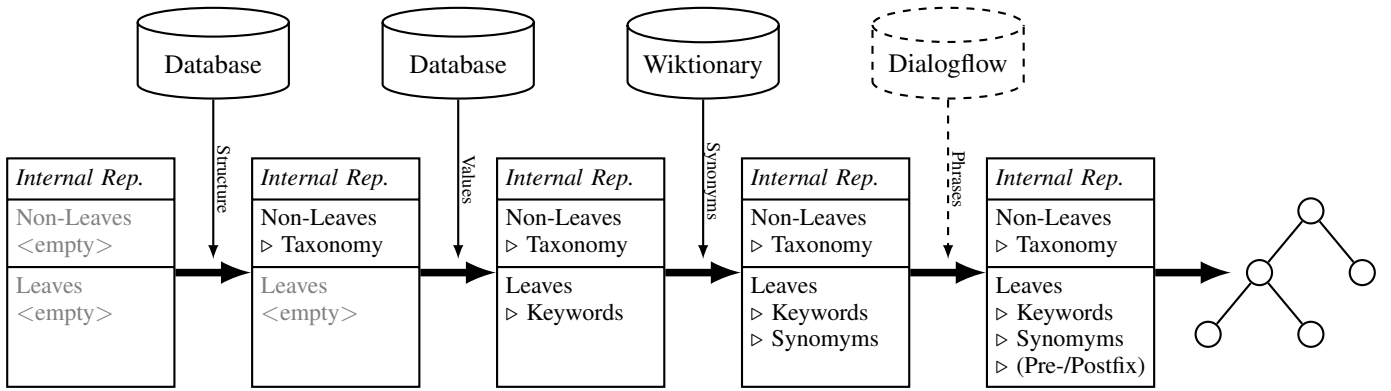


Fig. 2. The process to automatically create AOs. We use an (initially empty) internal representation that is successively populated from different sources: database structures and values, synonyms from Wiktionary, and – optionally – sample utterances from Dialogflow or similar sources.

ThingSystem. The virtual assistant translates natural language queries to the specialized programming language *ThingTalk*. The code is executed on *ThingSystem*, which creates service calls to the respective providers. Developers contribute to Thingpedia, a crowd-sourced public knowledge base of open APIs and their natural language interfaces. They specify trigger-actions such as If-This-Than-That (IFTTT) structures. The virtual assistant uses trigger-actions during code generation. The evaluation of the prototype shows that about 40% of tasks provided by a user familiar with the system are understood by Almond.

C. Natural Language Interfaces to Databases

Natural language interfaces to databases (NLIDB) have been studied since the late sixties; the associated conference (NLDB) is in the 24th iteration. Androutsopoulos et al. [5] give an introduction to the research area. Pazos R. et al. [6] review the state of the art. In the following, we briefly present representatives for common approaches.

Many NLIDB systems use specialized grammars. Rao et al. [7] use a *semantic grammar* to translate natural language to SQL queries. With customized production rules SQL queries are directly derived from the words in the utterances. Some NLIDB systems employ intermediate representations to grasp the intent of the natural language query. C-Phrase [8] uses an intermediate representation based on first order logic supplemented by additional higher-order predicates. It uses synchronous context-free grammars and lambda calculus expressions to convert natural language queries into the intermediate representation. Then, the intermediate representation is converted to an SQL query. More recent NLIDB approaches use machine learning techniques. Neelakantan et al. [9] use neural networks to map from language to SQL. Zhong et al. [10] employ reinforcement learning to improve quality.

All the above need developer-generated information: sample sentences, specialized grammars, or rules sets. We derive AOs directly from the database with minimal human effort.

IV. AUTOMATIC GENERATION OF ACTIVE ONTOLOGIES

We aim to create CIs for virtual assistants, chatbots, and the like (semi-)automatically. This way, we lower the effort for service providers to make their data accessible through a natural language interface. The sole precondition for our approach is that the service provider stores all information it wants to provide through the CI in databases. For example, if a service provider wants to publish information about touristic attractions it must store addresses, ratings, and the like of restaurants, museums, or galleries in accessible databases.

As underlying technology we use Active Ontologies. Usually, AOs are built manually. However, we have shown that it is possible to create AOs (semi-)automatically from web forms [11], [12]. In this work, we leverage the information provided by databases.

In contrast to web forms, databases always provide values, i.e. instances of concepts. We use the database values and synonyms to raise the linguistic competence of our AOs. Moreover, if sample utterances are present, we are able to add even more linguistic competence through an automatic extraction of common phrases. Our AOs are able to react to previously unseen user requests. Furthermore, they reply to composed requests.

Figure 2 shows an overview of our approach. We use an internal representation that is populated step by step. First, we extract the database structure and infer the concept hierarchy. Then, we create basic leaf nodes from the values in the database tables. Next, we add synonyms from Wiktionary, where applicable. If utterance samples are present, we add common phrases to the respective concepts. Finally, we generate the Active Ontology from the internal representation.

In the upcoming subsections we first describe the extraction of the concept hierarchy of the AOs from database schemes (subsection IV-A). Then we show how we increase the linguistic competence, i.e. how we add the leaf nodes to the AOs (subsection IV-B). Finally, we discuss the limitations of the approach, i.e. what a developer must review or add to the automatically generated AOs (subsection IV-C).

A. Taxonomy Extraction from Database Structures

AOs are hierarchic domain models. Inner nodes typically join information (gather nodes) to create a more complete view to the user’s request. We observed that database schemes follow the same intuition. For example, a database table *restaurant* may store information about *addresses*. The *addresses* again may be composed of a *ZIP*, a *city*, and a *street*. The *restaurant* table itself maybe used in different contexts. Thus, the *restaurant* table defines a concept that includes the sub-concepts *address*, *ZIP*, *city*, and *street*. For our prototype we use deductive databases.⁹ However, our approach can also be applied to relational databases as both database types store data in a structured way.

The structure is used to create the concept hierarchy of the Active Ontology, i.e the hierarchy of inner nodes. The database type only affects the way the database structure is extracted. Deductive databases contain triples that consist of an internal ID, the name of the property, and the property value. A property value may refer to another internal ID. For example, information about a restaurant is represented as follows:

```
[id01, @type, restaurant]
[id01, name, The Golden Eagle]
[id01, address, id01.address]
[id01.address, city, Karlsruhe]
[id01.address, ZIP, 76131]
```

We collect all property names and create a concept for each. Through the references in property values we infer hierarchies. For the above example the hierarchy in Figure 3 arises. Concept nodes can either be converted into gather or selection nodes. For our prototype we decided to only create gather nodes, because this is the best choice in most cases. Even though we found that for some concepts selection nodes would be the better choice, we were not able to come up with a generic rule to make this decision.

B. Leaf Node Generation

Now that we have created the hierarchy of concepts we need to extend the AO with linguistic competence. Up to now, the AO can join information only (and create a service call). However, it can not gather any information from a natural language request at all. Therefore, we create leaf nodes that match certain words or phrases in the utterance. We connect these leaf nodes to the respective inner nodes. For example, to create a leaf node that recognizes restaurants by their names

⁹A deductive database is a database equipped with a rule set. The rules are written in dialog, a simplified variant of logic programming [13], [14]. The deductive component of the database deviates additional knowledge from the data via rule executions. Queries are also composed in datalog. Ramakrishnana and Ullman describe deductive database systems as, “[...] database management systems whose query language and (usually) storage structure are designed around a logical model of data. As relations are naturally thought of as the ‘value’ of a logical predicate, and relational languages such as SQL are syntactic sugarings of a limited form of logical expression, it is easy to see deductive database systems as an advanced form of relational systems. [15]”

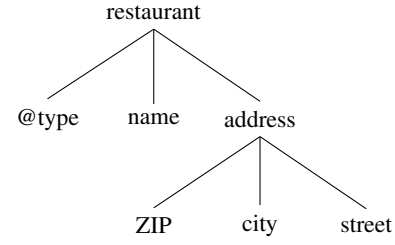


Fig. 3. A hierarchy extracted from database property names.

we can use the list of restaurants obtained from the database and attach it to the inner node *name*.

Besides vocabulary nodes, we generate all kinds of common node types (see section II), e.g. pre- and postfix nodes and specialized nodes such as date nodes. We use information from different sources to create leaf nodes. Next, we describe the sources, which kind of information we extract, the strategies to create leaf nodes, and which type of leaf nodes we create respectively.

1) *Database Values*: We create vocabulary, regex, and date nodes from database (property) values. For all string-valued properties we create vocabulary nodes. We join all values that belong to the same property (name). Thus, all names of restaurants are joined in a single vocabulary node. For numeric database values, e.g. phone numbers, ZIP codes, lengths, and the like, we use regex nodes. This way, we are able to extract any numeric information from the natural language input. Finally, to cope with specific phrases that contain date or time information such as “the day after tomorrow” we use date nodes. Date nodes recognize such phrases in the input and transform them into a machine-readable format.

2) *Lexical Databases and Dictionaries*: To amplify the linguistic competence of Active Ontologies one can add synonyms from lexical databases or dictionaries such as WordNet [16], [17], Wiktionary, or OpenThesaurus¹⁰. For our prototype we used Wiktionary for two reasons. First, as the values in our test databases are from Austrian service providers we need to recognize German utterances; Wiktionary provides the most extensive collection of German synonyms. Second, synonyms listed in Wiktionary are of high quality.

To retrieve synonyms, we first query Wiktionary for synonyms of the respective word. For each synonym we again look for synonyms. If no synonyms are available, we extract the so-called *similar words*. For each similar word we also search for synonyms. The list of synonyms and similar words is stored in a leaf node. Note that the leaf node does not pass the synonym to its parent nodes. Instead, the value of the associated property is used. For example, the leaf node that is supposed to recognize the word “Gaststätte” (German for “restaurant”) and all its (German and English) synonyms, e.g. “inn”, “restaurant”, or “eatery” passes the value *Gaststätte* to its parent nodes even if it recognizes the word “restaurant”

¹⁰OpenThesaurus: <https://www.openthesaurus.de>

in an utterance. This also relieves us from the issue that some values are German and others are English.

3) *Utterance Samples*: So far the generated AOs are capable to match keywords (or phrases) and their synonyms; they additionally detect dates and other regex-specified values. However, some user queries do not mention keywords directly. For example, a user might ask, “Where can I get pasta and tiramisu?” The utterance neither mentions restaurants directly nor the keyword “address”. However, obviously the user is interested in the address of an Italian restaurant. To overcome this limitation we automatically add common phrases that hint at a concept. Technically, such phrases form pre- and postfix nodes. We extract the phrases from sample utterances.

We are aware that sample utterances are not available for all domains/services. Thus, this step is optional. Alternatively, a developer can manually add pre- and postfix nodes to the AO.

For our prototype we use sample utterances from Dialogflow agents. Therefore, we extract all sample utterances from an agent and use the *intent* and *entities* as labels. With the help of the intent we are able to discover the appropriate AO part, e.g. restaurant requests. The entities have been linked to the respective database properties by the developer of the Dialogflow agent. Thus, we can determine the according concept in the AO. Additionally, often synonyms for entities are given. We add these to the synonym vocabulary nodes attached to the respective concept.

To extract pre- and postfixes we consider all entities in an utterance. We use all words preceding an entity as prefix and all succeeding as postfix. Of course, we stop discovery at the next entity. We consolidate all pre- and postfixes from all utterances for the same entity. Then we create one pre- and one postfix node per entity and attach them to the respective concept node. Given the Dialogflow sample utterance, “[Find me a [place to sleep]_{ent:type} in [Lisbon]_{ent:city}]_{int:hotel}”, we can identify the synonym *place to sleep* for the concept *@type* of the hotel AO part. Additionally, we extract the prefixes *find me a* (concept *@type*) and *in* (concept *city*) as well as the postfix *in* (concept *@type*).

C. Limitations

To automatically generate AOs, we have to make design decisions. For example, since we cannot automatically decide between selection and gather nodes, we create gather nodes for all concepts. However, in rare cases selection nodes are more appropriate. A developer can determine which node type is most suitable and select it accordingly. Another design decision concerns database values. We create word lists (for vocabulary nodes) for all string-valued properties. However, regex nodes might be more suitable to extract particular words or phrases, e.g. sub-strings.

In some cases, the extracted pre- and postfixes are either too specific or not specific enough. Modifying the phrases could improve matching with utterances. Entirely missing prefixes cause false positive matches and consequential conflicting hypotheses. Therefore, we expect that adding missing pre- and postfixes improves the accuracy of the AOs considerably. The

same applies to synonyms; a manual review improves accuracy since an automatic extraction of synonyms is error-prone.

V. EVALUATION

We evaluate our approach in three domains: tourism, hotels, and webcams. For each we have a deductive database and Dialogflow agents. The tourism dataset comprises information about local restaurants, ski rentals, events and the like. The hotel and web cam datasets contain information about the names and locations of hotels and web cams. In total, we included 3,861 data elements: 1,873 elements from the tourism, 1,303 from hotel, and 685 from web cam dataset.

We generate AOs for each domain in different variants. The basic AOs are created from the databases and synonyms only. All other configurations use sample utterances from Dialogflow agents to add pre- and postfix nodes. We use phrases from individual domains and all possible combinations of domain. For each domain we use all available samples provided by the agents for phrase extraction.

Note that any AO is enriched with pre-/postfixes where applicable. In other words, if we extract phrases from the web cam domain only but the other domains share concepts (e.g. locations) the respective phrases are added to all AOs. With that, we assess the impact of phrase extraction (the more the better?) and synergy effects (can we employ phrases from other domains?).

The database values are a mixture of English and German words. However, the sample utterances are in German. During AO generation we translate database values in the synonym mapping step (see subsection IV-B2). The generated leaf nodes are defective in some cases due to incorrect translations. However, we cannot measure the effect (if existing).

To assess the quality of our generated AOs we compare them to the Dialogflow agents. Therefore, we match the replies for a request returned by the AOs against Dialogflow. We assume that replies given by the Dialogflow agents are always correct, since all sample utterances have been manually annotated with intent and entity labels by developers. The Dialogflow agents contain 1,652 sample utterances (tourism: 1,140, hotels: 378, web cams: 134) with 4,597 entities (tourism: 2,430, hotels: 727, web cams: 1,440). To limit the effort, we analyzed a randomly sampled subset. We used 100 test utterances per domain for the configuration with no sample utterances, i.e. AOs built from databases and synonyms, and for the configuration with samples from *all* domains. For the other configurations we used 50 samples each.

The results of our study are depicted in Table I; the highest values for each configuration and per measure are highlighted. The first column shows the configuration for the phrase extraction, i.e. from which domain we took samples during AO generation: (T)ourism, (H)otel, and (W)eb cam. For all configurations we determine accuracy, recall, precision, and F_1 for the test utterances for individual domains (T, H, and W) and all domains (\forall). Note that we determine accuracy on a per-reply level, i.e. we count only answers that exactly match the Dialogflow result as correct. For precision, recall, and F_1

TABLE I
EVALUATION RESULTS FOR DIFFERENT PHRASE EXTRACTION SETTINGS. WE EITHER USED NONE, PHRASES FROM TOURISM (T), HOTEL (H), AND WEBCAM (W) AGENTS, OR VARIOUS COMBINATIONS.

Phrase Extr.	Accuracy				Recall				Precision				F ₁			
	T	H	W	∅	T	H	W	∅	T	H	W	∅	T	H	W	∅
none	0.11	0.00	0.16	0.09	0.12	0.06	0.29	0.18	0.34	0.60	0.79	0.58	0.18	0.11	0.42	0.27
T	0.30	0.02	0.22	0.18	0.50	0.22	0.29	0.35	0.72	0.45	0.42	0.55	0.59	0.30	0.34	0.40
H	0.14	0.00	0.06	0.06	0.04	0.10	0.26	0.17	0.13	0.40	0.75	0.48	0.06	0.16	0.39	0.25
W	0.14	0.02	0.24	0.13	0.19	0.31	0.51	0.34	0.42	0.63	0.84	0.66	0.26	0.42	0.63	0.45
T+H	0.26	0.04	0.22	0.17	0.43	0.25	0.55	0.28	0.71	0.47	0.45	0.52	0.55	0.35	0.37	0.36
T+W	0.32	0.02	0.40	0.24	0.44	0.33	0.55	0.46	0.78	0.52	0.96	0.79	0.57	0.40	0.70	0.58
H+W	0.02	0.14	0.34	0.16	0.18	0.24	0.55	0.33	0.26	0.50	0.78	0.51	0.21	0.32	0.65	0.43
T+H+W	0.30	0.08	0.28	0.22	0.45	0.35	0.50	0.45	0.74	0.62	0.79	0.78	0.54	0.43	0.61	0.57

we compare each element (i.e. *entity* in Dialogflow, *concept* in the AO) of the service call separately.

If an element is identified correctly, it is considered a true positive. Elements that were extracted mistakenly, i.e. they do not match an element extracted by the Dialogflow agent, are false positives. Any missing elements account for false negatives. Assuming that the service call elements for the exemplary request, “Find me a hotel in Lisbon,” are:

- Dialogflow: [hotel]_{ent:type}, [Lisbon]_{ent:location}
- Active Ontology: [hotel]_{ent:type}, [sauna]_{ent:feature}

In this example, *hotel* is a true positive, *Lisbon* a false negative, and the *sauna* accounts for a false positive.

The results indicate that our approach is feasible. Using the databases and synonyms for values only we achieve a precision of 58% (recall 18% and F₁ 27%). However, the accuracy (9%) shows that there is still much room for improvement. The evaluation also shows that enriching the AOs with sample utterances improves the quality in almost all cases. The best accuracy (40%), recall (55%), precision (96%), and F₁ (70%) are achieved for web cam requests when we use sample utterances from both the tourism and web cam domains. This clearly shows that phrases from other domains improves the linguistic competence of AOs. The effect can be further assessed with the results from the tourism domain with phrases extracted from the web cam domain (row *H*). Accuracy, recall, and precision improve in comparison to the configuration without any phrase extraction. This is due to shared concepts in both domains and similar sample utterances. For example, our approach extracts the prefixes *are there* for the concept @type and *in* for location from the web cam sample: “Are there [live pictures]_{ent:type} in [Salzburg]_{ent:location}?”¹¹ The prefixes can be applied to requests from the tourism domain such as, “Are there ski schools in Seeberg?”¹²

However, extracting sample utterances from the hotel domain often degrades the results as one can see, e.g. in the rows *H* and *H+W*. This is due to requests with many enumerations that occur frequently such as, “I am looking for a designer hotel with free parking, sauna, whirlpool, wifi, tennis court,

and a restaurant voucher.”¹³ Pre- and postfixes extracted from such samples produce both false positives and false negatives. For example, if the sample does not have an entity annotation for *free parking* our approach extracts the prefix *with free parking* for the concept *feature* (instance *sauna*) that causes false positives. Vice versa, if *free parking* has an annotation our approach misses the (correct) prefix *with* as we discontinue phrase extraction at the next annotation.

The best results for the hotel domain are achieved if we extract phrases from the web cam domain (see rows *W*, *T+W*, *T+H+W*). The results for tourism and web cams both improve when phrase samples from their own domain are used.

Accuracy values are low. However, using any kind of sample utterances increases the accuracy in all domains in almost all cases and reaches 40% in the best case.

We identified three errors classes that primarily affect the results. Rare errors are false positive elements due to missing pre-/postfixes. Concepts never mentioned in sample utterances are still part of the AO. These concept nodes have vocabulary, regex, or date nodes attached. Therefore, they might match requests even though the request have different intents. For example, a hotel might have a laundry but no sample mentioning it. The hotel AO then has a vocabulary node that reacts to any “laundry” in user requests. This may create false positives from requests such as, “I’m looking for laundries.”

Another error class is the incorrect selection of service calls. Our approach consolidates all concepts from each domain in a single AO. This way, we assure that only one service call is created from all hypothesis (combinations of elements). In some cases correct hypothesis have lower confidences than incorrect ones. If so, the root node chooses the wrong hypothesis and creates an incorrect service call.

Missing or incorrect synonyms form the third error class. Missing synonyms cause false negatives, while incorrect synonyms may produce false positives. An example for the latter is the German synonym *Haufen* (engl. *pile/bunch*) for the word *Berg* (engl. *mountain*) that is often used in colloquial idioms. This causes the AO part responsible for information on ski tours (to particular mountains) to react to phrases such as “ein Haufen Leute” (engl. “a bunch of people”).

¹¹Original: “Gibt es [Live Bilder]_{ent:type} in [Salzburg]_{ent:location}?”

¹²Original: “Gibt es Skischulen in Seeberg?”

¹³Original: “Ich suche ein Designerhotel mit gratis Parkplatz, Sauna, Whirlpool, WLAN, Tennisplatz und einem Restaurantgutschein.”

VI. CONCLUSION AND FUTURE WORK

We have presented an approach to automatically create virtual assistants from databases. Our virtual assistants use Active Ontologies to model the problem domain and process user utterances.

To generate AOs we first infer a concept hierarchy from database schemes and create word lists and regular expressions from extracted database values. We extend the word lists with synonyms from Wiktionary. Additionally, if sample utterances are present, we build pre- and postfix nodes to enhance the linguistic competence of our AOs.

Our evaluation in the domains tourism, hotel, and web cams shows that our approach successfully generates AOs that extract relevant information from user utterances. When we use sample utterances, precision and recall increases. Our approach successfully learns phrases from a domain (e.g. tourism) and uses it for queries from another (e.g. hotel).

We plan to improve the AO generation process. For the time being, we create one AO per domain. We experiment with smaller AOs, that may extract intents with higher confidences. This also evades the duplication of subtrees (e.g. for addresses of restaurants and cinemas).

Another improvement involves the general structure of the AOs. In some cases multiple pre- and postfix nodes fire at the same time, which makes intent extraction ambiguous. An additional layer of selection nodes that selects the most reasonable pre-/postfix may be beneficial.

We observed that some phrase segments are more meaningful than others. Therefore, we have implemented an alternative to pre-/postfix nodes. Instead of using full pre-/postfix phrases we extract single words and create a vocabulary node for each. We attach all of them to an additional layer of gather nodes that represent the original pre-/postfixes. This way, we create subtrees that capture bags of words. We calculate the TFIDF for each word per intent (each intent is a document and all intents correspond to the document set). The TFIDF value is used as confidence for the new vocabulary nodes.

A first case study shows the potential of this approach. We used 239 sample utterances from Dialogflow agents for the intent “request opening hours”. Additionally, we created 13 synthetic requests for addresses (e.g., *What is the address of the restaurant The Toothless Shark in Cologne?*).

We created bag-of-words subtrees for both intents and integrated them into the AO. On the test set the AOs answered opening hours request with an accuracy of 65% and addresses with an accuracy of 23%. The low accuracy for the latter is due to the small number of sample utterances.

With the bag-of-words subtrees the AOs can extract multiple intents from single user utterance. For the time being, virtual assistants such as Dialogflow cannot combine different intents. Thus, our approach improves the state of the art. Composed requests such as, “Give me the address and the opening hours of the hotel Tivoli Oriente in Lisbon,” achieve an accuracy of 40%. In the future, we plan to further investigate the bag-of-words-approach and evaluate it on a larger data set.

VII. ACKNOWLEDGEMENT

The evaluation of our approach was partially supported by Onlim GmbH¹⁴ that provided the data for testing. We thank Dr. Ioan Toma (Onlim GmbH) who greatly assisted our research.

REFERENCES

- [1] D. Guzzoni, C. Baur, and A. Cheyer, “Active: A Unified Platform for Building Intelligent Web Interaction Assistants,” in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology - Workshops, Hong Kong, China, 18-22 December 2006*. IEEE Computer Society, Dec. 2006, pp. 417–420.
- [2] D. Guzzoni, “Active: A unified platform for building intelligent applications,” PhD Thesis, École Polytechnique Fédérale De Lausanne, Jan. 2008.
- [3] A. Cheyer and D. Guzzoni, “United States Patent: 8677377 - Method and apparatus for building an intelligent automated assistant,” USA Patent 8 677 377, Sep., 2005.
- [4] G. Campagna, R. Ramesh, S. Xu, M. Fischer, and M. S. Lam, “Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant,” in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 341–350.
- [5] I. Androutsopoulos, G. Ritchie, and P. Thanisch, “Natural language interfaces to databases – an introduction,” *Natural Language Engineering*, vol. 1, no. 01, pp. 29–81, Mar. 1995.
- [6] R. A. Pazos R., J. J. González B., M. A. Aguirre L., J. A. Martínez F., and H. J. Fraire H., “Natural Language Interfaces to Databases: An Analysis of the State of the Art,” in *Recent Advances on Hybrid Intelligent Systems*, ser. Studies in Computational Intelligence, O. Castillo, P. Melin, and J. Kacprzyk, Eds. Springer Berlin Heidelberg, 2013, no. 451, pp. 463–480.
- [7] G. Rao, C. Agarwal, S. Chaudhry, N. Kulkarni, and D. S. Patil, “Natural language query processing using semantic grammar,” *International journal on computer science and engineering*, vol. 2, no. 2, pp. 219–223, 2010.
- [8] M. Minock, “C-Phrase: A system for building robust natural language interfaces to databases,” *Data & Knowledge Engineering*, vol. 69, no. 3, pp. 290–302, 2010, special Issue: 13th International Conference on Natural Language and Information Systems (NLDB 2008) – Five selected and extended papers.
- [9] A. Neelakantan, Q. V. Le, M. Abadi, A. McCallum, and D. Amodei, “Learning a natural language interface with neural programmer,” *arXiv preprint arXiv:1611.08945*, 2016.
- [10] V. Zhong, C. Xiong, and R. Socher, “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning,” *CoRR*, vol. abs/1709.00103, 2017.
- [11] M. Blersch and M. Landhäuser, “Easier: An Approach to Automatically Generate Active Ontologies for Intelligent Assistants,” in *Proceedings of the 20th World Multiconference on Systemics, Cybernetics and Informatics (WMSCI 2016)*, Orlando, FL, USA, Jul. 2016.
- [12] M. Blersch, M. Landhäuser, and T. Mayer, “Semi-automatic Generation of Active Ontologies from Web Forms for Intelligent Assistants,” in *Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, ser. RAISE ’18. Gothenburg, Sweden: ACM, 2018, pp. 28–34.
- [13] S. Ceri, G. Gottlob, and L. Tanca, “What you always wanted to know about Datalog (and never dared to ask),” *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 146–166, Mar. 1989.
- [14] M. Krötzsch, S. Rudolph, and P. H. Schmitt, “A closer look at the semantic relationship between Datalog and description logics,” *Semantic Web*, vol. 6, no. 1, pp. 63–79, 2015.
- [15] R. Ramakrishnan and J. D. Ullman, “A survey of deductive database systems,” *The Journal of Logic Programming*, vol. 23, no. 2, pp. 125–149, 1995.
- [16] G. A. Miller, “Wordnet: A lexical database for english,” *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995. [Online]. Available: <http://doi.acm.org/10.1145/219717.219748>
- [17] C. Fellbaum, *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

¹⁴Onlim GmbH: <https://www.onlim.com>

GADIS: A Genetic Algorithm for Database Index Selection

Priscilla Neuhaus, Julia Couto, Jonatas Wehrmann, Duncan D. Ruiz and Felipe Meneguzzi

School of Technology, PUCRS - Pontifícia Universidade Católica do Rio Grande do Sul - Porto Alegre, Brazil

Email: [priscilla.neuhaus, julia.couto, jonatas.wehrmann]@edu.pucrs.br, [duncan.ruiz, felipe.meneguzzi]@pucrs.br

Abstract—Creating an optimal amount of indexes, taking into account query performance and database size remains a challenge. In theory, one can speed up query response by creating indexes on the most used columns, although causing slower data insertion and deletion, and requiring a much larger amount of memory for storing the indexing data, but in practice, it is very important to balance such a trade-off. This is not a trivial task that often requires action from the Database Administrator. We address this problem by introducing GADIS, A Genetic Algorithm for Database Index Selection, designed to automatically select the best configuration of indexes adaptable for any database schema. This method aims to find the fittest individuals for optimizing both query response time, and disk required for the indexed data. We evaluate the effectiveness of GADIS through several experiments we developed based on a standard database benchmark, compare it to three baseline indexing strategies, and show that our approach consistently leads to a better resulting index configuration.

Index Terms—Database, Indexing, Artificial Intelligence, Genetic algorithms, Learning system.

I. INTRODUCTION

Creating indexes is the main action to improve database query performance [7, Chap. 15], since the indexes are the most used technique to speed up queries response [9]. However, it is important to properly choose the columns to be indexed, given that it also affect time to insert and update data and increase disk consumption. The Database Management Systems (DBMS) optimizer is responsible for analyzing queries and choosing the most efficient way to access information. The goal of an optimizer is to find options for running a given query and evaluate the cost of each choice, so that the chosen one would provide the best performance for retrieving the data.

Achieving the best indexing configuration for a database is not a trivial task [6]. Ideally, all the frequently queried columns should be indexed for a faster data retrieval. However, it is quite complex to find a balanced trade-off between performance and storage required. It is quite often the case when the cost-based optimizer is not able to find a proper solution, requiring the DBA to make a final decision on the database architecture regarding indexing strategies.

In this paper we developed GADIS, an approach based on Genetic Algorithms (GA) to help finding an optimal index configuration. We use two fitness functions: 1) an optimization objective to maximize the database performance considering INSERT, DELETE and SELECT queries; and 2) designed to

optimize the query response time by search for faster index configurations when compared to the initial one.

We perform a set of experiments using TPC-H, a well known database benchmark [11], and we compare our results with three baseline approaches: (i) the initial index configuration; and (ii) indexes derived from a random-search algorithm. Results show that GADIS outperforms the baselines, allowing for better index configuration on the optimized database. Finally, we observe that our approach is much more prone to find a proper solution that the competitive methods.

II. BACKGROUND

A. Genetic Algorithms

Holland [3] developed the concept of Genetic Algorithm inspired by the evolutionist theory. GA simulates the principles of biological evolution by repeatedly modifying a population of individual using rules modelled on reproduction and gene combinations. In this simulation of evolution, it leverages the best genes from the fittest individuals of a population to continue across the next generations. GA models individuals in terms of their genome, typically represented as strings of bits. Each generation replaces the previous population by its fittest *offspring*, where fitness is computed by a fitness function that assigns a score to each individual. Fitness typically measures how well an individual solves the problem at hand. The algorithm initializes with a random population and works through *selection*, *crossover*, and *mutation*. That process continues until the optimize criterion is satisfied or a certain number of generations is reached. Due to its random nature, GA improves the chances of finding a global solution.

B. TPC-H Benchmark

TPC¹ is a a non-profit corporation that produces database benchmarks. We chose TPC-H because it models a business database having realistic ad-hoc queries. TPC-H has a database schema, a workload and performance metric tests. The database size varies according to a constant named scale factor (SF). The workload is composed of 22 queries of varying complexity, and 2 refresh functions, that simulate data insertions and deletions in the two larger tables (`orders` and `lineitems`). In the performance test, the benchmark executes a power test and then a throughput test. The power

TABLE I
GA DEFINITIONS APPLIED IN THE CONTEXT OF THE DBMS

GA Definition	DBMS Application
Gene	0 if the column is not indexed; 1 otherwise
Individual	a database state represented by a vector that refers the columns of the schema
Population	collection of database states
Parents	two database states selected to be combined and next create a new database state
Mating pool	a collection of parents that are used to create the next population
Fitness	a function that tells us how good each database state is by running the benchmark
Mutation	a way to introduce variation in our population by randomly swapping the genes (0 - 1) of two individuals
Elitism	a way to carry the best individuals into the next generation

test (POWER@SIZE), calculates how fast the system computes answers to single queries. It executes a function to insert data, then it runs all queries in parallel, then it executes another function to delete data. The throughput test (THROUGHPUT@SIZE) measures how many queries were executed in the elapsed time for parallel query streams to simulate multiple users. It computes the ratio between the total number of queries and the total time spent to run the queries. TPC-H also presents the query-per-hour metric (QPHH), obtained from the geometric mean of power and throughput test. It captures the overall performance level of the system, both for single-user and multi-user mode.

III. METHOD

We already saw that indexes can speed up the data access. However, when we create or delete indexes we must verify which combination of indexes is the best one for queries selection. Specifically, for optimizing the TPC-H database, there are 2^{45} possible combinations for column indexing. Our approach is based on GAs trained directly on a running database, designed to evolve individuals that represent the whole index structure (i.e., all the columns on the database). We map GA definitions to the DBMS context in Table I.

A. Individual Representation

We use a straightforward individual representation that is based on binary vectors. In this strategy, each vector position denotes whether a column is indexed or not. Formally, we represent an individual as a binary vector \mathbf{x} , where $|\mathbf{x}| = C$ and C is the number of all columns in the whole database schema. Hence, x_i denotes whether the i^{th} column should be indexed by the DBMS. Naturally, both primary and foreign keys are always indexed, and therefore not affected by any crossover, mutation or additional random-based action on \mathbf{x} .

The initial population with n individuals is randomly created by sampling bits from an uniform distribution. Each bit corresponds to a specific gene having two possible actions: *create* or *drop* one index. Each individual is a concatenation of the binary representation of all columns from all tables. TPC-H contains 61 columns across 8 tables, with 16 primary

and foreign keys indexed by default. From the remaining columns, only 24 are used on the benchmark queries. Hence, each individual is comprised by $C = 24$ mutable genes, which generates a search space containing $2^{24} = 16,777,216$ possible combination of indexes.

B. Fitness Function

During the evolutionary process, we use a fitness function to estimate the degree of adaptation to the environment for each individual in the population. We first use the QPHH as fitness function that aims to optimize the performance for running queries while being computationally cheaper for data insertion and removal. We also propose a simpler approach, which optimizes the speed-up time for running all the 22 select queries in the benchmark.

More specifically, we want to find an individual represented by a genome that is capable of achieving high QPHH, but using the very least memory as possible. Formally, let $\mathcal{Q}(\mathbf{x})$, $\mathcal{H}(\mathbf{x})$ and $\mathcal{P}(\mathbf{x})$ be the functions that estimate the QPHH, THROUGHPUT@SIZE and POWER@SIZE for a given \mathbf{x} . Note that the estimate of $\mathcal{P}(\mathbf{x})$ is calculated by running all the queries in the benchmark, including delete and insert ones that are quite slow in heavily indexed databases. Thus, a database in which most of the columns are indexed would most likely yield lower POWER@SIZE values. The QPHH-based fitness function would be hereby referred as $\mathcal{Q}(\mathbf{x})$, calculated by Eq. 1.

$$\mathcal{Q}(\mathbf{x}) = \sqrt{\mathcal{P}(\mathbf{x}) \times \mathcal{H}(\mathbf{x})} \quad (1)$$

Our second fitness function optimize the time performance for running the SELECT queries in the benchmark. In this case, we want to find individuals that are capable of retrieving data efficiently, without considering data insertion and removal. This is achieved by optimizing the speed-up of the current individual when compared to a baseline one, namely, the initial state of the database. In this case, we refer to the function that calculates the total query time for a given individual as $\mathcal{T}(\mathbf{x})$. Finally, the speed-up-based fitness function is given by

$$\mathcal{S}(\mathbf{x}) = \frac{\mathcal{T}(\mathbf{x}_I)}{\mathcal{T}(\mathbf{x})} \quad (2)$$

where \mathbf{x}_I denotes the individual for the initial state of the database. Therefore, individuals with lower values of $\mathcal{T}(\mathbf{x})$ have higher fitness values than those that take more time to run the benchmark. In this case, we are necessarily optimizing the database storage requirements for index data. In summary, the proposed fitness functions are two-fold: (i) $\mathcal{Q}(\mathbf{x})$, which optimizes the performance for INSERT, DELETE and SELECT commands, and by transitivity, it also helps to lower memory requirements for indexes; and (ii) $\mathcal{T}(\mathbf{x})$ that directly optimizes the running time for all queries.

C. Selection

Our approach uses the popular and effective tournament method as selection technique, developed by Horn et al. [8]. It is a strategy for selecting the fittest candidates from the current generation in a GA. The process initiates with two candidate

points selected randomly from the current population, that compete for survival in the next generation. The next step is to compile points randomly to compose a tournament set, where each member is compared with other members. We need to specify the size of the tournament set as a percentage of the total population. Hence, the tournament set size implies the degree of difficulty in surviving. If the tournament size is larger, weak candidates have a smaller chance of getting selected as it has to compete with a stronger candidate. We use the selection pressure to determine the rate of convergence of the GA. This is a probabilistic measure of a candidate likelihood of participation in a tournament. Here, the convergence rate is proportional to the selection pressure, and the GA is capable of identifying optimal or near-optimal solutions over a wide range of selection pressures. The tournament selection works either for positive or negative fitness values.

D. Crossover and Mutation

After obtaining the individual from selection method, we apply the crossover genetic operator on population. In this phase, two individuals exchange information to produce the offspring. Both crossover and mutation occur only with respect to some probability previously defined. The main goal of the use of crossover is to make it possible for the genes of two individuals to generate an improved individual. Higher crossover values lead to in-depth exploitation of the current population individuals, but constraining the exploration of the search space. We employ the two-point operator [10], that determines two random crossover points to mark at which points of the two parents will occur the split. Next, the tails of their two parents are swapped to get a new offspring, which would integrate the population. It is important to define the crossover probability (cp), which controls the frequency of the application of the crossover operator on the individuals. For instance, when using $cp = 1.00$, crossover is applied over the entire population. On the other side, when $cp = 0.00$, the entire new generation is made from exact copies of individuals from old population (which can suffer mutation as well).

The next step is the mutation operator, that introduces random changes into the characteristics of the individuals. Mutation plays a critical role in GA, as crossover leads the population to converge by making the individuals in the population alike. Mutation reintroduces genetic diversity back into the population. In this phase, we need to set the mutation probability and the mutation rate. The first parameter sets the chance of each individual to be mutated, whilst the second one refers to the number of genes that would be changed.

Frequently, the mutation rate is rather small and depends on the length of the individuals. Therefore, new individuals produced by mutation will not be very different from the original one. Since we set each individual as a bit vector, the mutation operator is responsible to change the bit value of 1 to 0 or vice-versa. We also provide some experiments using a linear decay for the mutation probability. Such a decay allows for larger exploration during the initial generations, increasing

the genetic diversity, while allowing for in depth exploitation during the latter stages of the optimization procedure.

IV. EXPERIMENTAL SETUP

We carried out all experiments in hardware with Intel(R) Xeon(R) Platinum 8175M CPU @ 2.50GHz, 30GiB RAM and Non-Volatile memory controller (SSD), running Ubuntu Linux 16.04. We used DEAP [2], an open-source Python package, to implement the genetic algorithm. To evaluate all approaches, we use the TPC-H benchmark with scale factor of 1GB. We run the performance test and calculate the $POWER@SIZE$ and $THROUGHPUT@SIZE$ metrics for each execution of TPC-H. Then we calculate the QPHH for each individual ($Q(x)$).

A. Baselines

For evaluating our approaches we defined three main baseline strategies for database indexing. We compared our approach with the following strategies: (1) TPC-H initial state (only primary and foreign keys are indexed); (2) results achieved via a random-search algorithm that optimizes the proposed fitness functions. We run the random search using the same number of individuals as in our optimization algorithm.

B. Evaluation Measures

In our experiments, we run the final evaluation using a different set of query streams to the database, in order to make sure that our GA was not overfitting the training examples. First, we set the best indexing configurations found during the whole training phase, in order to run the complete benchmark. For quantitative evaluation, we used QPHH and the storage required by the index data (in MB).

C. Parameters

The first three experiments were designed to optimize the fitness function $Q(x)$, in order to maximize the QPHH metric. The two latter ones focused on minimizing the second fitness function, namely the benchmark runtime given by $T(x)$. For all experiments we used a initial population of 50 individuals, evolved during 50 generations, with mutation rate of 0.05 and elite size of 4. Parameters such as fitness function, mutation probability (mp), crossover probability (cp) and mutation decay (md) are defined as follows: $GADIS-[Q, mp=0.9, cp=0.8, md=0.05]$ and $GADIS-[T, mp=0.9, cp=0.6]$. The complete training procedure for each of those experiments (including random search) took about a week.

V. RESULTS

Figure 1 shows the performance of our approaches during the training phase. The chart shows values of QPHH as fitness for all evolved individuals; and the second chart depicts values of time required to run the test benchmark (only with SELECT queries), by using our second fitness function, namely $T(x)$. GADIS have shown to be much better in terms of consistency, once it can find several good database configurations with ease when comparing to the baseline. It is clear that both $GADIS-[Q]$ $GADIS-[T]$ present similar performance during

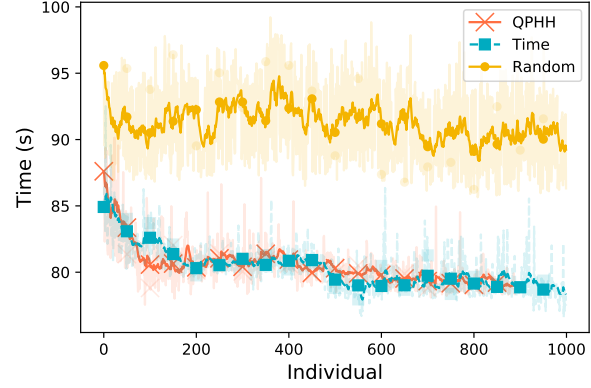
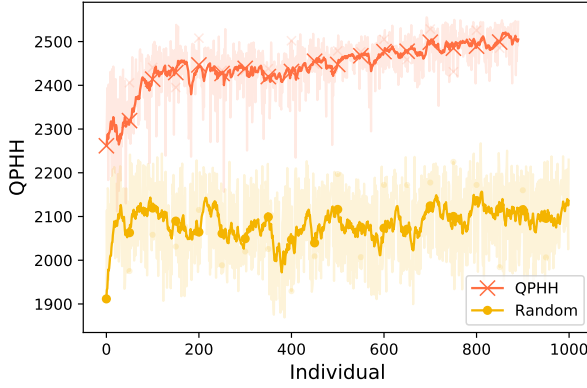


Fig. 1. QPHH and Time during training procedure.

TABLE II
RESULTS IN TERMS OF QPHH, TIME AND INDEX SIZE

Methods	QPHH	Time	Index Size
Initial State	1678	149.3s	599 MB
Random Search	1864	145.3s	1196 MB
GADIS-[Q]	2631	71.2s	1193 MB
GADIS-[T]	3077	60.6s	1600 MB

the training phase, though the latter presents itself as a much faster approach.

Table II shows quantitative results in terms of QPHH, Time and Index Size for each one of the approaches. Note that the best performing approach as measure in QPHH and Time is achieved by GADIS-[T]. Additionally, both versions of GADIS were able to achieve top results in all used metrics. GADIS-[Q] also requires roughly the same disk space when compared to the baseline, though with 40% better QPHH performance.

VI. RELATED WORK

In the past years, other researchers have used GA to improve database performance. Korytkowski et al. [4] present an automatic way to find the best set of indexes for a database, using GA. Different from our approach, their fitness function is based on time spent in insert operations and a single query, and they make experiments with just one table.

Pedrozo et al. [5] modelled an index tuning architecture applied to hybrid storage environments using GA to create indexes in the DBMS. They also use TPC-H benchmark to evaluate their approach, but unlike us, they did not apply the performance test provided by the benchmark.

Boronski et al. [1] propose a model to optimize the response time of a set of queries by creating indexes in a relational database. Unlike us, they use the response time of each group of queries to measure their performance and then compare it with the Oracle advisor.

VII. CONCLUSIONS

In this paper we developed a Genetic Algorithm-based approach for automatic index selection in databases called GADIS. This approach can find database configurations that outperform all the baselines in most of the scenarios, while saving storage requirements. We have observed that the training procedure of GADIS is very consistent, which allows us to find proper database configurations within only a few generations. In addition, GADIS is easily suited to be implemented on any database system. The main limitation of our work is that to find a good solution one has to run several distinct database configurations and evaluate them by using a benchmark. Such a procedure is somewhat costly, and required about a week of processing to optimize the indexes for the used database. For future work, we plan to improve GADIS so we can learn general rules for database agnostic indexing using metadata rather than performing a per-database optimization.

REFERENCES

- [1] R. Boronski et al., "Relational database index selection algorithm," in *CN*, A. Kwiecień et al., Eds. Springer, 2014, pp. 338–347.
- [2] F.-A. Fortin et al., "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.
- [3] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- [4] M. Korytkowski et al., "Genetic algorithm for database indexing," in *ICAISC*, L. Rutkowski et al., Eds. Springer, 2004, pp. 1142–1147.
- [5] W. G. Pedrozo et al., "An adaptive approach for index tuning with learning classifier systems on hybrid storage environments," in *HAIS*. Springer, 2018, pp. 716–729.
- [6] E. Petraki et al., "Holistic indexing in main-memory column-stores," in *SIGMOD*. ACM, 2015, pp. 1153–1166.
- [7] R. Ramakrishnan et al., *Database Management Systems*, 3rd ed. McGraw-Hill, Inc., 2003.
- [8] J. rey Horn et al., "A niched pareto genetic algorithm for multiobjective optimization," in *CEC*, vol. 1, Citeseer. IEEE, 1994, pp. 82–87.
- [9] P. Rob et al., *Database Systems Design, Implementation and Management*, 5th ed. Course Technology Press, 2002.
- [10] W. M. Spears et al., "An analysis of multi-point crossover," in *Foundations of genetic algorithms*. Elsevier, 1991, vol. 1, pp. 301–315.
- [11] A. Thanopoulou et al., "Benchmarking with TPC-H on off-the-shelf hardware," in *14th International Conference on Enterprise Information Systems*. Springer, 2012, pp. 205–208.

Research on Page Object Generation Approach for Web Application Testing

Yimei Chen, Zheng Li, Ruilian Zhao and Junxia Guo*

College of Information Science and Technology,
Beijing University of Chemical Technology, Beijing, China

*Corresponding: gjxia@mail.buct.edu.cn

Abstract—Test code generated by using the page object design pattern during web testing is easy to maintain. Page clustering is an essential stage of the page object approach. However, existing methods only consider the DOM structure in page clustering, which leads to inaccuracy when generating page objects. A state with the same DOM structure may result in an entirely different migration. The method of considering only the DOM structure cannot accurately generate page object classes. In order to improve the accuracy of page object generation, this paper not only considers DOM structure information but also considers CSS styles and the attributes of DOM elements in page clustering. Based on the experimental evaluation results, our method can automatically generate page objects that cover most of the application functions, which is more effective for the creation and maintenance of web test cases.

Index Terms—Web applications, Automated testing, Page object.

I. INTRODUCTION

As an essential stage of software development, software testing can be described as the process of identifying the correctness of programs. The primary purpose is to find possible errors in programs [1]. The cost of software testing is about 50% of entire software development cycle [2]. With the faster step of business, the software development cycle also becomes shorten and shorten. As a result, software testing needs to be finished in limited time, especially for quick-updated web applications. Because of cost constraints, software companies want to test their software as quickly as possible. Thus, research on automated testing has received significant attention in both industry and academia. Automated testing can run frequently, shorten the test cycle, quickly respond to changing requirements, and make the development process more agile.

The existing end-to-end automated test tools have a similar problem of maintaining test scripts during software evolution. Although those test tools can make testing easier, they cannot help testers write well-structured test scripts. To adapt to changing web applications, testers need to change test suites, which represents a considerable workload. Thus, the techniques for both test suite generation and maintenance are needed.

The page object pattern [3] is an effective mode on enhancing test suite maintenance and can reduce code duplication. A page object is an object-oriented class that acts as an interface

to the web page of applications under test. Individually, all element attributes and element operations of a page object are encapsulated in a class. Whenever testers need to interact with elements of the user interface, the test case will use the approaches of the page object class. The test code is separated from the page elements and its methods of operation, in order to reduce the impact of changes in page elements on the test code. If page elements are changed, we only need to modify the code of the corresponding page object (the corresponding class) without modifying the test code. Therefore, it is beneficial to adopt the page object mode in maintaining web test suites.

However, the existing tools based on page object mode have their limitations. Only the DOM state is used for page clustering, which leads to the inaccuracy of generating page objects. Two web pages with the same DOM structure can show different visual effects and trigger completely different events, which should be divided into two page objects. To address this problem, we propose an approach for automatically generating page objects for web application testing, which is more accurate and efficient.

In this paper, we conduct in-depth research on page object generation and use it in web testing. Its main contributions are listed as follows.

1. Oriented to the automated generation of page object, we theoretically analyze influence factors related to page clustering. Based on the consideration of the DOM structure of two pages, we also consider CSS styles and attributes of DOM elements. According to the multiple influence factors, we propose a two-stage clustering algorithm.
2. We implement a prototype tool for automatically generating test cases for web applications, which combines multiple influence factors we propose in page clustering. In this way, we effectively solve the problems of repeated inaccurate testing in the automated testing process and improve the efficiency of the test.
3. We do a series of experiments with several kinds of web applications to evaluate the approach of this paper.

The organization of this paper is as follows. Section II presents the motivation. Section III introduces our approach, theoretically analyzing influence factors on page clustering and also proposes a clustering algorithm of multiple influence fac-

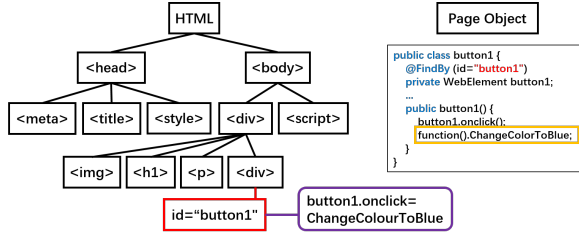


Fig. 1. A simple DOM tree of pageA.

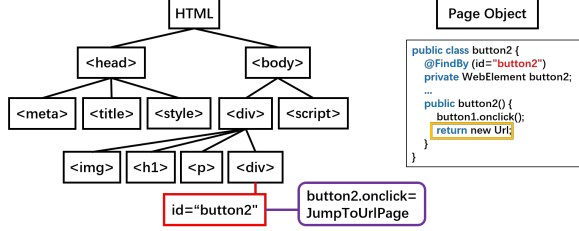


Fig. 2. A simple DOM tree of pageB.

tors. Section IV presents the experimental results to assess the effectiveness of our approach and its limitations. The relevant research is presented in Section V. Section VI concludes and presents the future works.

II. MOTIVATION

Generally, generating page objects starts from a state-based model (graph) of the web application. The model consists of nodes and edges, where nodes are dynamic DOM states of web pages and edges are attribute-based transitions between nodes. The information of nodes and edges can be crawled from the web application. Then, the similar web pages are grouped into an abstract representation by the page clustering algorithm. Consequently, the related interactions between the web pages may change according to the clustering results. Finally, a set of page objects is generated for the abstract web pages and their interaction.

From the above analysis, we can find that the accuracy of page clustering is a crucial aspect when generating page objects. Page clustering relies on the similarity between web pages. However, the existing studies focus on structural features only, such as tag frequency, URL, DOM elements and so on, ignoring CSS styles and the attributes of DOM elements, to measure the similarity between web pages. This leads to the inaccuracy of page clustering and page object generation.

As an example, assume that two web pages with the same DOM structure can show different visual effects and trigger completely different events, which should be grouped into two page objects. However, the existing tools which only use the DOM structure information for clustering web pages will classify these two pages into one category, causing the inaccuracy of page objects. For instance, as shown in the Fig.1 and Fig.2, different attribute(i.e. id) values are set to the <div> tags with the same DOM structure. Meanwhile, different attribute handlers *button1* and *button2* are set to

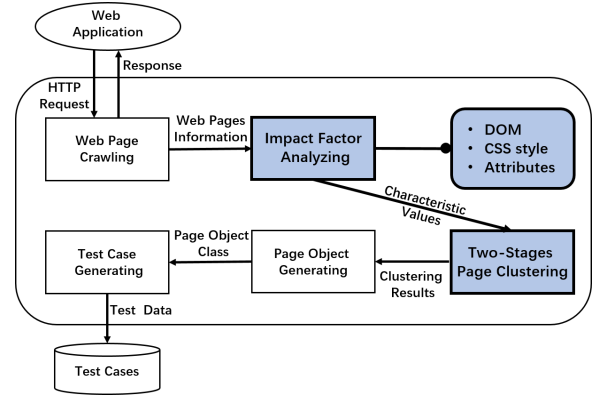


Fig. 3. An overview of the approach.

different ids. Even if their tags and structures are identical, pageA and pageB represent different states, and they should be divided into two page objects.

For measuring the similarity between web pages, the features which are using in existing approaches are insufficient. To solve this problem, we conduct an in-depth analysis of page information and raise a novel set of features considering the structure, CSS styles and attributes of pages, as well as the corresponding similarity measurement to distinguish pages with the same DOM structures but different functions.

III. OUR APPROACH

A. Approach Overview

The overview of our approach is shown in Fig.3. This paper design an automated testing framework which has five main steps. In the first step, by using a specific crawler, we can simulate user actions to get information. Then, we theoretically analyze factors that affect page clustering, including DOM structures, CSS styles and attributes of DOM elements. In this step, we get their respective characteristic values. In step three, according to the two-stages clustering algorithm mentioned in this article, web pages are clustered. Next, page objects can be generated for every clustered page. Finally, test cases can be generated by calling page elements in corresponding page object classes.

B. influence factors related to page clustering

1) *influence factors of DOM structure*: DOM is a neutral interface between platform and language that allows programs or scripts to dynamically access content, styles and structure of updated documents [4]. The DOM structure represents the hierarchical structure of a web page and is very important for quantifying the similarity degree of web pages. In this paper, we use tree edit distance (TED) [5] to calculate the similarity of DOM. The Tree Edit Distance is the minimum number of tree editing operations that convert tree T to tree T'.

2) *influence factors of CSS styles*: In web applications, HTML tags not only display information about the structure and content of the page but also show some information about performance. Cascading Style Sheets (CSS) define how HTML

elements are displayed. Web developers nowadays put the information not only in the HTML tag hierarchy but also in the style sheets. CSS styles of web pages are also important information for page clustering.

In CSS, each style is usually named by a class value. The HTML elements in a web page are rendered by referring to the class values. The HTML elements with the same class value have the same style. Therefore, we consider using information on class attributes as a kind of assist for page clustering when generating page objects. This may avoid the misclustering case where the DOM structure of two pages is the same, but different CSS styles are used, which should not be classified into one category.

The process of building a style matrix between two pages is as follows. Firstly, we parse the HTML document of the web page and get a collection of all class values used in this page. We can record the class value information of the two web pages into two sets A and B. Then we can use Jaccard distance to calculate the distance between the class value matrix of two pages. The calculation method is listed in Equation 1.

$$D_j(A, B) = 1 - J(A, B) = \frac{A \Delta B}{|A| + |B| - |A \cap B|} \quad (1)$$

When $A = B$, $D_j(A, B) = 0$. The smaller the value, the more similar the two pages are. We use the calculation result of Jaccard distance as the style matrix values. According to the above steps, the style matrix values of any two pages can be respectively calculated and stored in the corresponding vectors.

3) *influence factors of the attributes of DOM elements*: If the structure of the two pages is the same, but the attributes of DOM elements bound on the nodes are different, different functions will be triggered. So the attributes of DOM elements may be helpful in clustering. HTML pages contain different HTML elements which may have different attributes and different functions. If all the attributes are taken into account, it will not only significantly increase the complexity of the method, but also reduce the accuracy of comparison, for example, the src attributes of nodes.

The id value in a web page is a kind of unique identifier for a DOM node. Usually, scripting languages use id as a tag to find the node where id is located. So id value is significant for migration between states. We can distinguish different events bound to a node by id. Therefore, we consider that id value is a useful attribute in clustering and state migration.

We use Jaccard distance of id values to construct an attribute-based matrix between two pages. We get a collection of all the id values of a web page and use Equation 1 to calculate the id-distance of two web pages.

4) *Using tag filter to reduce the DOM state*: There are a large number of tags in HTML pages. Different tags have different effects. Not all tags in the HTML document are helpful in page clustering. We propose a tag filter method to reduce the pending tags in the HTML document, which focus on the effective tags and improves the usability and accuracy of the discovered tags. So that in page clustering, tag filter

can remove some interference, which can improve accuracy and speed.

In HTML pages, tags like <head>, <style>, <script> and <link> contain data that is not displayed to the user as content, which is a factor in clustering with DOM structures. In theory, by removing these tags that are not used for structure analysis, not only will the DOM structure will become smaller and simplified, but also the consumption time will be reduced. DOM is the foundation of web application display. The more accurately the page structure is analyzed, the more completely its functions are understood. Based on the DOM structure, tag filter optimizes the clustering approach and indirectly improves the accuracy of subsequent operations.

C. A two-stages clustering algorithm using multiple influence factors

The factors affecting page clustering mentioned above have their pros and cons. They have different effects on different types of websites. If we only use one of them to measure all types of web pages, there will be inaccurate classification problems that affect the accuracy of page object generation.

Algorithm 1 A two-stages clustering algorithm using multiple influence factors

Input: HTML pages P_n crawled by crawler

Output: A set of clustering results S_j of HTML pages

```

1: Get  $P'_n$  by using tag filter on  $P_n$ 
2: Calculate DOM tree edit distance matrix  $M_{DOM}$  of  $P'_n$ 
3: Generate  $S_i$  using hierarchical clustering on  $M_{DOM}$ 
4:  $n$  = number of crawled HTML pages in  $P'_n$ 
5:  $i$  = number of clustering results in  $S_i$ 
6:  $j$  = number of clustering results in  $S_j$ 
7: for all  $(s_1, s_2, \dots, s_i) \in S_i$  do
8:   if Consider CSS styles then
9:     Get class values sets  $S_{CSS}$  of  $P'_n$ 
10:    Calculate  $M_{CSS}$  of  $S_{CSS}$  based on Equation 1
11:    Generate  $S_j$  using hierarchical clustering on  $M_{CSS}$ 
12:    return  $S_j$ 
13:   else if Consider attributes then
14:     Get id values sets  $S_{id}$  of  $P'_n$ 
15:     Calculate  $M_{id}$  of  $S_{id}$  based on Equation 1
16:     Generate  $S_j$  using hierarchical clustering on  $M_{id}$ 
17:     return  $S_j$ 
18:   else if Consider CSS styles and attributes then
19:     Calculate  $M_{Cid}$  based on Equation 2
20:     Calculate  $M_{Cid}$  based on Equation 1
21:     Generate  $S_j$  using hierarchical clustering on  $M_{id}$ 
22:     return  $S_j$ 
23:   else
24:     return  $S_i$ 
25:   end if
26: end for

```

Therefore, we propose a two-stage clustering algorithm using multiple influence factors as shown in Algorithm 1. We use tag filter on the pages crawled for preprocessing

TABLE I
TEST SUBJECTS.

Application	states	URLs	edges	DOM length
Aminer	12	4	17	58.714 kB
Termonline	57	3	56	26.194 kB
Musicbible	27	7	42	185.73 kB
BUCT	13	5	12	46.917 kB
DBLP	8	8	14	31.698 kB
CBA	17	17	51	31.156 kB
Chinacoop	55	49	108	18.26 kB
Xinxishibao	49	15	48	196.542 kB
Wanshifu	35	27	79	49.965 kB

to simplify the DOM structure. In the first stage, we use hierarchical clustering to cluster pages based on the DOM tree edit distance matrix of two pages. According to the clustering results of the first stage, we re-cluster the pages that are grouped into the same category. In the second stage, we consider different influence factors for clustering, CSS style and id attribute. Finally, the clustering results are returned according to different factors. This algorithm can correctly classify most different types of websites with better accuracy.

For any two web pages obtained by the crawler, the matrix M_{CSS} based on the influence factor of CSS styles can be obtained by calculating Jaccard distance of class value sets based on Equation 1 and the attribute matrix M_{id} can be gotten by computing Jaccard distance of id value sets. The final matrix value M_{Cid} of multiple influence factors according to a certain weight is calculated using Equation 2.

$$M_{Cid} = \omega \times M_{CSS} + (1 - \omega) \times M_{id} \quad (2)$$

ω and $(1 - \omega)$ respectively represent the weight of the matrix value of CSS styles and the attributes of DOM ids. Setting reasonable weights in the final calculation is necessary. In order to determine the value of the above two weight parameters, this paper performs parameter adjustment experiments on several web applications.

IV. CASE STUDIES

In order to study the effectiveness of page object generation approach proposed in this paper, we selected thirty web applications in six fields as test objects, which contains the fields of information search, arts, portals, sports, governments and life services. Due to the limited space, we chose one or two of each category for display. In this section we list the result of nine web applications, including Aminer, Termonline, Musicbible, BUCT, DBLP, CBA, Chinacoop, Xinxishibao and Wanshifu. Then we use a two-stage clustering algorithm using multiple influence factors on these applications and analyze its impact on the accuracy of page object generation.

A. Test subjects

Table I gives the information about the nine experimental objects, including each application's name, the number of states, the number of visited URLs, the number of edges and the average length of the DOM.

TABLE II
COMPARISONS OF EIGENVALUES OF USING TAG FILTER OR NOT.

HTML	Tag Fliter	The Value of Eigenvector
index	N	0,731,0,697,719,734,742,772
	Y	0,727,0,693,715,730,738,766
state6	N	731,0,731,151,79,65,80,1195
	Y	727,0,727,151,79,65,80,1184
state14	N	697,151,697,0,128,145,147,1151
	Y	693,151,693,0,128,145,147,1140
state23	N	772,1195,772,1151,1183,1192,1211,0
	Y	766,1184,766,1140,1172,1181,1200,0

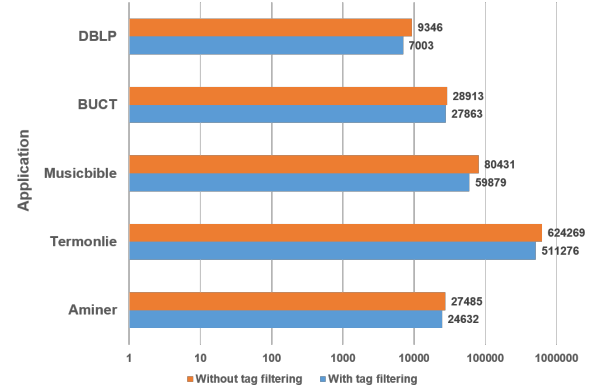


Fig. 4. Comparison of time consumption (in ms) in generating DOM-based feature matrix with or without tag filter.

B. Experimental results

1) *Research on the effectiveness of tag filter on DOM states reduction:* We compare page objects generated by using tag filter or not. We find that the page states generated before and after tag filter are the same.

Table II is the data of feature vector based on the DOM tree edit distance extracted from test subject DBLP. The first column shows the labels instead of different HTML pages. The second column shows whether tag filter is used to reduce DOM states. The remaining columns are the experimental value of the feature vector.

In Table II, the size of the number represents the degree of difference between DOMs. By observing experimental data, we find that tag filter can simplify DOM structure and make it more similarity.

Fig.4 shows a comparison of the time consumption of generating a DOM-based feature matrix using tag filter or not. It shows that the times using the tag filter are shorter.

Therefore, We can say that using tag filter will not only produce more reasonable page objects but also reduce time consumption. So we use tag filter to simplify DOM structure before generating a feature matrix.

2) *Research on the effectiveness of two-stages clustering algorithm using multiple influence factors and weight setting:* We compare generated page objects with different influence factors for test subjects and manually analyze page objects generated of each web application. The analysis results are used as criteria for judging the effects of each influence factor.

TABLE III
COMPARISON OF THE EFFECTS OF VARIOUS INFLUENCE FACTORS.

Application	DOM	DOM+CSS	DOM+ID
Aminer	1.52%	0.00%	0.00%
Termonline	0.88%	0.88%	0.00%
Musicbible	6.55%	6.55%	5.13%
BUCT	12.82%	0.00%	12.82%
DBLP	14.29%	0.00%	14.29%
CBA	13.97%	6.62%	8.82%
Chinacoop	14.41%	0.00%	5.39%
Epaperxxsb	0.00%	1.70%	1.70%
Wanshifu	2.52%	2.18%	1.01%
total	66.95%	17.93%	49.15%

Table III shows the result of the two-stage clustering algorithm proposed in this paper with the percentage of the mis-classified page combination number vs. the total page combination number. The first column is web applications under test. Second column shows the results that only using the DOM structure for page clustering. The third column shows the results that consider the DOM structure and CSS style for page clustering. The fourth column shows the results using DOM structure and attribute influence factors. The last line is the sum of the error rates for each method on all test subjects.

Through the data in Table III, we can find that for the application Aminer, the result of page objects generated when considering CSS style and DOM attribute for page clustering is the same as the result of the manual analysis. Moreover, they are both better than only considering the DOM structure. For the application Termonline, Musicbible and Wanshifu, the results that use DOM attribute have the lowest error rate when generating page objects. For the application BUCT, DBLP, CBA and Chinacoop, page objects generated by DOM structure and CSS styles have the lowest error rate. In addition to application Epaperxxsb, results of CSS-assisted DOM or ID-assisted DOM are better than using only DOM.

Therefore, we conclude that the generating accuracy of page object generated using our clustering algorithm is better than the method that only considering the DOM structure. However, the results of DOM structure with CSS styles and DOM structure with attribute have different performance for different applications. Therefore, we also do experiment that combine those two influence factors with different weights to achieve the best classification accuracy.

To set reasonable weights for CSS styles and DOM attributes in the final calculation, we did experimental research. In addition, we need to find the suitable condition for starting the second stage. If the result of first stage shows that the two pages are quite different, the second stage need not to be started. Therefore, We set the trigger threshold of the second stage in page clustering to be 0.5. When the distance between two pages exceeds the threshold in first stage, the clustering algorithm will start the second stage's processing.

The weights of the CSS style and DOM attribute values are set based on Equation 2. Then we raise the threshold of starting the second stage and adjust the weight of two influence factors. Through manual analysis, we get a reasonable threshold and

TABLE IV
COMPARISON OF EFFECTS OF DIFFERENT WEIGHTS.

Threshold	$\omega=0.2$	$\omega=0.4$	$\omega=0.6$	$\omega=0.8$
0.5	6.55%	6.55%	6.55%	6.55%
0.6	6.55%	6.55%	6.55%	6.55%
0.7	6.55%	6.55%	6.55%	6.55%
0.8	6.55%	6.55%	6.55%	6.55%
0.9	6.55%	6.55%	6.55%	3.13%

weight set. The result of the manual analysis is used in our experiments.

The experimental results are shown in Table IV. Here we show the result of test subject Musicbible. The first column is the threshold of starting the second stage in page clustering. It is incremented from 0.5 to 0.9 to determine which weight has the best effect. The lines show the weight of CSS style influence factors based on Equation 2. The numbers in the table indicate the pages that are the percentage of the mis-classified page.

When the threshold of starting second stage is 0.9 and the respective weights of M_{CSS} and M_{id} are 0.8 and 0.2, the result of generating page object is most similar to the result of manual judgment results. Therefore, the best effect of two-stages clustering algorithm using multiple influence factors is to take 0.9 as the threshold of starting the second stage in page clustering, with the weight of 0.8 for M_{CSS} and 0.2 for M_{id} .

V. RELATED WORKS

There are a number of testing approaches for web applications. Some of those approaches are suitable for the early days of Web applications, while others are more suitable for solving modern web technologies, such as AJAX and node.js [6], [7]. Some technologies are based on service-oriented framework, for example research [8]. Some use web page information extraction techniques. As a commonly used approach, the DOM similarity is used to judge the location of the required information, and the location information can be extracted directly without being disturbed by the noise information [9].

Web applications are different from traditional application software. web applications use BS structures and communicate with servers through browsers. However, local applications can run off the network. The difference is that web applications can only be run in various forms of browsers. In the traditional test [10], the web test is divided into two different test forms: black box and white box. Researchers use the corresponding models, such as control flow graphs or navigation maps, to conduct web application system testing studies. For example, Arcuri A [11] proposes a automated white-box testing method, in which test cases are generated automatically using evolutionary algorithms. Tests will be rewarded based on code coverage and fault finding metrics. Binkley D proposed a functional road map for testing web applications [12]. Liu X combined statement-based fault classification with spectrum-based software fault location in order to improve the accuracy of fault location and provide more possible fault information

for programmers [13]. In the research of Milani and Mirzaaghaei [14], by mining the human minds contained in the manually written test cases, they generate test cases for those points that are not found in the program under test. However, when the same function has multiple possible outcomes, a considerable amount of redundant test cases are generated. In the research of Yu Bing [15], a method based on a page object model for automatically generating web application testing is proposed. In this approach, the same function has multiple possible results, resulting in a significant probability that the generated page object is repeated, and finally, a redundant test case is generated.

When facing massive data information, it is possible to obtain incomplete or irrelevant information even through search engines, but with the development of web data mining, the problem is alleviated [16], [17]. Web data mining analyzes web page content to measure the importance of one page. Many algorithms have been proposed to extract content using the DOM tree. For example, in the method proposed in the research of Elyasov A [18], the DOM of the page is analyzed as an input to propose a testing framework of Javascript evolutionary. In the method proposed by Pagi V B [19], by using the embedded semantic tree kernel they can extract opinion content from web pages. In order to address repairing Internationalization Presentation Failures problems in web pages, Mahajan S [20] uses clustering to group stylistically similar elements in a page. It then performs a guided search to find suitable CSS fixes for the identified clusters.

Web pages contain a considerable amount of content that is not related to the web page theme. When acquiring information, this irrelevant information will affect efficiency and accuracy. It is often necessary to do some preprocessing on the web page. Uma [21] proposes an approach that cleans up and displays important information from web pages in standard format by eliminating noise and using unsupervised technology.

VI. CONCLUSIONS AND FUTURE WORKS

The existing generated page object tools only consider the DOM structure as the influence factor, which causes incorrect classification. In order to solve this problem, this paper theoretically analyzes the influence factors, which considers not only DOM structure, but also CSS styles and the attributes of DOM elements, and uses tag filter to simplify DOM structure. We propose a two-stage clustering algorithm using multiple influence factors when generating page objects. Therefore, the problem of incorrect classification of page objects is optimized. Moreover, we implement a prototype tool to automatically generate test cases for web applications to improve test efficiency based on page object. With experimental research, we find a suitable weight set for influence factors and threshold and verify the validity of our approach.

In future, we would like to analyze more factors. For example the type of events in client side scripts, which can handle different triggerable events. Moreover, in order to get more available information, we may use machine learning to

determine whether state and function are related. Besides, we will extend tested objects into other types of applications.

ACKNOWLEDGMENT

The work described in this paper is supported by the National Natural Science Foundation of China under Grant No.61702029, No.61672085 and No.61872026.

REFERENCES

- [1] Offutt J, Ammann P.: Introduction to Software Testing. Cambridge University Press (2008)
- [2] DENG Zhidan, YANG Haiyan, WU Ji.: Test data generation and selection approach for Web application based on constraint-solving. *Computer Engineering and Applications* **52**(18), 214–221(2016)
- [3] PageObjects, <http://code.google.com/p/selenium/wiki/PageObjects>. Last accessed 13 Mar 2015
- [4] W3C, <https://www.w3.org/DOM/>. Last accessed 1 Jun 2019
- [5] Stefan Schwarz, Mateusz Pawlik, Nikolaus Augsten.: A New Perspective on the Tree Edit Distance. In: International Conference on Similarity Search and Applications 2017, LNCS, vol. 10609, pp. 156–170. Springer, Cham (2017).
- [6] Serdar Dogana, Aysu Betin-Cana, Vahid Garousia.: Web application testing: A systematic literature review. *Journal of Systems and Software* **91**(1), 174–201(2014)
- [7] Wang Lina, Li Huai, Zhao Lei.: Ajax Web automatic testing model based on simulation of users. *Journal of Huazhong University of Science and Technology* (2016)
- [8] Torsel A M.: A Testing Tool for Web Applications Using a Domain-Specific Modelling Language and the NuSMV Model Checker. In: Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, pp. 383–390.(2013)
- [9] Pan Xinyu, Chen Changfu, Liu Rong, Wang Meiqin.: Content extraction based on the similarity of the Web pages' DOM tree nodes path. *Microcomputer and its Applications* **35**(19), 74–77(2016)
- [10] Myers G J, Sandler C, Badgett T.: The Art of Software Testing. 2nd edn.(2004)
- [11] Arcuri A.: RESTful API Automated Test Case Generation. *ACM Transactions on Software Engineering and Methodology* (2019)
- [12] Binkley D, Ceccato M, Harman M.: Tool-Supported Refactoring of Existing Object-Oriented Code into Aspects. *IEEE Transactions on Software Engineering* **32**(9), 698–717(2006)
- [13] Liu X, Liu Y, Li Z.: Fault Classification Oriented Spectrum Based Fault Localization. *Computer Software and Applications Conference. IEEE* (2017)
- [14] MilaniFard A, Mirzaaghaei M, Mesbah A.: Leveraging existing tests in automated test generation for web applications. In: ACM/IEEE International Conference on Automated Software Engineering. ACM, 2014, pp. 67–78.(2014)
- [15] Yu B, Ma L, Zhang C.: Incremental Web Application Testing Using Page Object. In: Third IEEE Workshop on Hot Topics in Web Systems and Technologies. IEEE Computer Society, 2015, pp. 1–6.(2015)
- [16] ZHANG Nai-Zhou, CAO Wei, LI Shi-Jun.: A Method Based on Node Density Segmentation and Label Propagation for Mining Web Page. *Chinese Journal of Computers* **38**(2), 349–364(2015)
- [17] HUANG Yanjiao, WU Qin, LIANG Jiuzhen.: Boosted constrained conditional random fields for Web object information extraction. *Computer Engineering and Applications* **51**(23), 143–148(2015)
- [18] Elyasov A, Prasetya I S W B, Hage J.: Search-Based Test Data Generation for JavaScript Functions that Interact with the DOM. In: Memphis, U.S.A, proceedings of the, IEEE 29th International Symposium on Software Reliability Engineering.(2018)
- [19] Pagi V B, Wadawadagi R S.: Opinion Content Extraction from Web Pages Using Embedded Semantic Term Tree Kernels. *International Conference on Computational Intelligence and Data Engineering*, pp. 345–358(2018)
- [20] Mahajan S, Alameer A, Mcminn P.: Automated Repair of Internationalization Presentation Failures in Web Pages Using Style Similarity Clustering and Search-Based Techniques. In: 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST). IEEE Computer Society.(2018)
- [21] Uma R, Latha B.: Noise elimination from web pages for efficacious information retrieval. *Cluster Computing*, pp. 1-20(2018)

A Class-level Test Selection Approach Toward Full Coverage For Continuous Integration

Yingling Li^{*†}, Junjie Wang^{*†}, Qing Wang^{*†‡§}, Jun Hu^{*†}

^{*}Laboratory for Internet Software Technologies, [†]State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences, Beijing, 100089, China

[‡]University of Chinese Academy of Sciences, Beijing, 100089, China, [§]Corresponding author
email: {yingling, wangjunjie, wq, hujun} @itechs.iscas.ac.cn

Abstract—Continuous Integration (CI) is an important practice in agile development. With the growth of integration system, running all tests to verify the quality of submitted code, is clearly uneconomical. This paper aims at selecting a proper test subset towards full coverage of all changed and affected code so as to reduce the cost of CI testing. We propose FEST, a novel approach, which searches for the full dependencies of changed code at the class level and then selects test classes related to the changed and affected classes. We assess FEST from fault detection efficiency and cost effectiveness based on 18 open source projects with 261 continuous integration versions from Eclipse and Apache communities, and compare it with the state-of-the-art approach ClassSRTS (as baseline). Results show that FEST (1) can not only cover all faults detected by actual CI testing and baseline, but also find new faults in 25% and 18% versions respectively. (2) shows better or equal test scale benefits than actual CI testing (in 98% versions) and baseline (in 99% versions); and can compensate risk of omitting necessary tests for actual CI testing (in 62% versions) and baseline (in 73% versions).

I. INTRODUCTION

Continuous Integration (CI) is a widely-applied development practice which requires developers to integrate their code into the master codebases frequently. It can improve the productivity, facilitate fast feedback of quality problems, and help projects release more often [1], [2]. One of the best practices in CI is to make your build self-testing, called CI testing. Each programmer must do a complete build and run (and pass) all or eligible unit tests before submitting work [1], [3]. With the growth of integration system, running all tests to verify the quality of submitted code, is clearly uneconomical. The big challenge of CI testing is how to optimize the test set to reduce test size as much as possible without sacrificing quality. These are many test case selection techniques proposed to tackle this problem [2], [4]–[16].

Generally speaking, the methods of test case selection can be split into static and dynamic techniques. Dynamic selection technique collects test dependencies by dynamically running tests on the previous revision, and selects a test set that may reach the code changes [7]–[9], [15], [17]. However, dynamic test dependencies for large projects may be time-consuming to collect, and for real-time systems, dynamic selection techniques may not be applicable, because code instrumentation for obtaining dependencies may cause timeouts or interrupt

normal test run [5], [15]. Besides, for programs with non-determinism (e.g., due to randomness or concurrency), dependencies collected dynamically may not cover all possible traces, leading to omission of necessary tests [5], [6]. Static selection technique does not require to execute tests; it uses static program analysis to infer the dependencies between changed code, affected code and test code [4], [5], [13], [18], [19]. It is easier to manipulate than dynamic technique thus draws more and more attention. However, the drawback of existing static approaches is that the selected test sets either cannot fully cover the necessary tests or are too redundant due to inefficient static analysis techniques [4]–[6], [13], [20].

In this paper, we propose a novel test selection approach called FEST (i.e., Full dEpendency based class-level test SelecTion) to select a proper test subset for CI testing based on class-level static dependencies. FEST first locates the changed classes of submitted commits, and generates full dependencies of source code at the class level. It then searches a complete set of affected classes and identifies affected test classes. Finally, it extends the test classes based on recently failed tests.

We experimentally evaluate FEST from fault detection efficiency and cost effectiveness on 261 CI versions of 18 projects from two large open source communities (i.e., Eclipse and Apache). For comparison, we refer the real-world practice of CI testing as the *actual CI testing* and apply the state-of-the-art approach ClassSRTS [5], [13] as baseline. The results show that (1) FEST can cover all faults detected by actual CI testing and baseline, and find new faults in 25% and 18% versions respectively; (2) compared with baseline, FEST shows better or equal test scale benefits in 99% versions, and higher risk compensation in 73% versions.

The main contributions of the paper are as follows:

- We design a new class-level test selection approach (FEST) to select a proper test subset towards fully covering all changed code and affected code. It can resolve full dependency relations at the class level, which improves previous work by capturing the dependencies of hidden references; and design an algorithm to incrementally and iteratively search the affected tests.
- We conducted experiments on 261 integration versions of 18 open source projects to evaluate the fault detection efficiency and cost-effectiveness of our approach, and results are promising.

II. APPROACH

This paper proposes an approach, called FEST (i.e., Full dependency based class-level test SelecTion). There are four major steps in FEST (shown in Figure 1), the following subsections will explain the details of the four steps.

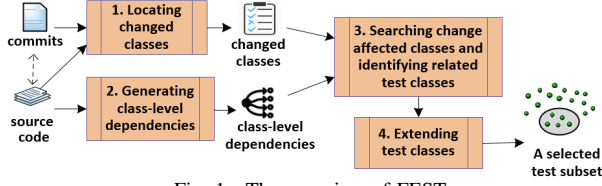


Fig. 1. The overview of FEST

A. Locating the changed classes of submitted commits

For each CI, we have three phases to analyze and locate the changed classes. Firstly, we look for the **names of changed files** from the log message of the commits. Secondly, we look for the **changed lines of each changed file** by applying “*git diff*”, and filter the blank lines and annotation lines. Thirdly, we locate the changed code on associated classes. In detail, an improved open source tool Doxygen (named as Doxygen#) is applied to analyze the structure and dependencies of source code. We further parse the *xml* files generated by Doxygen# to obtain the code structure information of each file, including class names, the start line and end line of each class, and annotation information of each method (e.g., *@Test*, *@BeforeClass*, *@Before*). Based on the code structure information, we map the changed lines to their corresponding classes, and then label them as changed classes. Note that the commits can involve both program code and test code. We also locate changed test classes for better selecting the related tests. After this step, FEST will output a set of changed classes (C_Set).

B. Generating the full dependencies of code version

To ensure the full dependencies of code captured, we refer to the relations defined in *UML*. There are six relations among classes or objects defined in *UML*. Table I presents these relations and their representation in code (refer Orso’s work [21]), as well as the corresponding relations in code. There are two categories of relations in code: inheritance and invocation [22]. Some of the invocation types can be directly obtained with code dependency analysis tools (e.g., Doxygen#), and we call it as direct reference (short for *R*). Other invocation types can not be directly obtained with existing tools, and we call it hidden reference (short for *HR*). These hidden reference relations are usually ignored in the existing approaches [5], [20] or tools for code dependency analysis (e.g., Doxygen) because they are hard to resolve. We carefully consider and resolve the relation.

For inheritance and reference relations of invocation category, we directly capture them using the tags (e.g., *derivedcompoundref*, *referencedby*) in *xml* files outputted by Doxygen#. For hidden reference, we parse the *xml* files based on its representation in code (as shown in Table I) and the code structure information of class files obtained in Step 1. Take the first *dependency* relation as an example (i.e., the invocation

TABLE I
MAPPING OF RELATIONS BETWEEN RELATIONS IN *UML* AND IN CODE

Relations in UML	Representation in code	Relations in code
Generalization	A subclass $e1$ extends its superclass $e2$.	Inheritance
Realization	Class $e1$ implements an interface $e2$	Inheritance
Dependency	A Method of Class $e1$ uses Class $e2$ as an argument, e.g., the invocation by “ <i>Class.forName(‘P.e2’)</i> ” in reflection	Invocation (HR)
	A Method of Class $e1$ uses Class $e2$ in a cast operation.	Invocation (HR)
	Class $e1$ instantiates Class $e2$ in $e1$ ’s methods, then invokes its methods or variables.	Invocation (R)
Association	Class $e1$ uses Class $e2$ as a member variable.	Invocation (HR)
	A method in Class $e1$ invokes methods or member variables of member Class $e2$.	Invocation (R)
	Class $e1$ defines a member variable, and initiates it by invoking Class $e2$ ’s methods or variables, or using $e2$ in a cast operation.	Invocation (HR)
Aggregation	Class $e1$ uses Class $e2$ as an argument to instance.	Invocation (HR)
Composition	If Class $e2$ is a component of Class $e1$, it can only be instantiated in Class $e1$.	Invocation (R)

by “*Class.forName(‘P.e2’)*” in reflection), for each *xml* file, we check whether there is such hidden reference relation by its representation in code. If there is “*Class.forName*” in the *xml* file, we regard it as this case, and obtain the information of the referenced class (i.e., $e2$ as the name of referenced class, P as the name of $e2$ ’s package), and record its position (i.e., line of code); then we map the position to corresponding class based on the code structure information, and obtain the reference class and its information (e.g., class name). Finally, the hidden reference can be built with the obtained information of reference and referenced classes.

For convenience of step 3, FEST outputs an inheritance relation graph (*HRG*) and an invocation relation graph (*VRG*).

C. Searching affected classes and identifying related test classes

In this section, we design a BFS-based (i.e., Breadth-First Search) search algorithm to incrementally and iteratively look for a complete set of affected classes and identify related test classes. Note that since our approach considers the class-level dependency relations, the *tests* in this paper means the test classes.

In detail, for each changed class C_i in C_Set , firstly, we use BFS to search its all subclasses from *HRG*, and all referenced classes from *VRG* which invoke class C_i or its subclasses, and add them to the set of affected classes A_Set . Then, for the changed class C_i or each affected class, we check whether it is a test class. Note that we also check the changed class C_i , which ensures to select the newly-added tests or changed tests. Following existing work [6], [20], if it contains at least one method (the method name started with “test” or the method has the annotation “*@Test*”, “*@BeforeClass*”, “*@Before*”), we regard it as a test class. If it is a test class, we will add it to the set of related tests T_Set . Finally, when this loop is over, and A_Set is not null, for each class a_i in A_Set , we apply BFS iteratively to search its subclasses and referenced classes until there is no new class found, and identify new test classes, add

them to T_Set . After the above process, we search affected classes and identify the set of related tests T_Set .

D. Extending test classes

In this section, we aim to complement some tests to re-detect the deferred faults. These faults detected in the previous versions might have not been fixed and deferred to subsequent versions. This strategy has also been applied in existing studies [2], [11], [23]. In detail, we look for the failed test classes in recent n versions, then check whether they run again. If not, we add them into T_Set . We have experimented n from 1 to 30, and results showed that the performance is better and more stable when n is 10. Hence, we apply the recent 10 versions in the following experiments. Finally, we obtain a final test set (T_Set).

III. EXPERIMENT DESIGN AND EVALUATION METRICS

A. Subject projects and data preparation

According to the activeness and CI testing history availability, we chose 12 Eclipse projects and 6 Apache projects to evaluate our approach presented in Section II. We collected the CI testing history from November 2017 to January 2018 for the chosen projects, which contains test classes, test methods, results, etc. From which we can obtain the failed test information for each CI version. Then we collected commits from the *Git* repository by executing “*git log*”, checked out source code by executing “*git checkout*”; and built the relationship of the three datasets by the commit *id* of CI versions. Finally, we chose the versions for our experiment, in which the number of changed classes and the number of executed CI tests are greater than 0; and obtained 261 versions from the 18 projects.

We ran all experiments on a 3.40 GHz Intel Core i7-3770 machine with 8 GB of RAM, running Ubuntu Linux 14.04.3 LTS and Java 64-Bit sever version 1.8.0_73.

B. Baseline approach

To further evaluate the performance of our approach, we take ClassSRTS [5], [13], the state-of-the-art class-level static selection technique, as baseline. Besides, we also compare FEST with the selected tests of actual CI testing, which are recorded in the repositories.

We did not select any dynamic approach as baseline because in the most recent work [5], ClassSRTS is compared to the state-of-the-art dynamic approach, and results show that ClassSRTS is comparable with the dynamic approach. The reason we did not compare with dynamic selection is because it needs to run tests to obtain the dependency information, which is costly and limits its realization in our study.

C. Evaluation metrics

Following existing work, faults denote failed test classes where a test class is regarded as fail if at least one of the test cases in the test class fails, otherwise, it is regarded as pass. For each CI version, we compare the selected tests and the failed tests in the CI testing history, and count the failed test classes contained in the set of selected tests to get the

TABLE II
BASIC METRICS USED IN THE STUDY

Metrics	Description
Ct	A set of actual CI tests (i.e., actual tests ran during CI testing).
St	A set of selected tests by FEST or ClassSRTS (marked by $St@F$ and $St@C$ respectively).
Cft	A set of faults detected by actual CI testing.
Tft	A set of total faults detected by FEST or ClassSRTS (marked by $Tft@F$ and $Tft@C$ respectively).

number of detected faults. Regarding a new test class which is not included in the current set of actual CI testing, it will be labeled as a detected fault if its first run in the subsequent versions fails by following previous work [24]. It is because it should be detected in the previous version, but it was omitted by actual CI testing. **To facilitate understanding, we present basic metrics used in this paper in Table II.**

1) *Fault detection efficiency*: In this dimension, we define two metrics to evaluate fault detection efficiency.

(a) **Fault coverage** ($Fcovg$) means the percentage of the faults found by test selection approach (i.e., FEST or ClassSRTS) compared with the faults detected in actual CI testing. Following existing work [4], [12], [20], [25], it is defined as Equation 1.

$$Fcovg = \frac{|Tft \cap Cft|}{|Cft|} \times 100\% \quad (1)$$

In it, Tft is the faults detected by the measured approach, it can be $Tft@F$ (for FEST) or $Tft@C$ (for ClassSRTS). This metric is also used to evaluate the coverage of the faults detected by FEST compared with the faults detected by ClassSRTS, where Cft will be replaced by $Tft@C$. When $|Cft|$ is 0, we treat $Fcovg$ as 100%.

(b) **Fault detection enhancement** ($Fenhm$) means the new faults which are detected by the measured approach compared with actual CI testing or ClassSRTS. It is defined as $NewFt$.

2) *Cost effectiveness*: In this dimension, we evaluate the cost effectiveness from test scale benefits and risk compensation ability.

(a) **Test scale benefits** ($Testscal$) evaluates the return on investment (ROI), i.e., the number of detected faults divided by the number of running tests, as the existing work [9], [26], [27]. As we could not obtain the exact set of all faults, we use the union set of the faults detected by all investigated approaches and actual CI testing, as previous work [12], [20]. Therefore, we apply a penalty coefficient to adjust ROI, which is the proportion of the number of faults detected by current approach to the number of all faults. Hence, $Testscal$ is defined as Equation 2. The higher value of $Testscal$ the better.

$$Testscal = \frac{|Tft|}{|St|} \times \frac{|Tft|}{|Cft \cup Tft@F \cup Tft@C|} \quad (2)$$

Here, Tft is the fault set of the measured approach, i.e., it can be Cft (for actual CI testing), $Tft@F$ (for FEST) or $Tft@C$ (for ClassSRTS). When $|St|$ or $|Cft \cup Tft@F \cup Tft@C|$ is 0, we simply treat $Testscal$ as 0.

(b) **Risk compensation ability** ($Riskcomp$): Running insufficient tests for CI could lead to omit some potential faults, while running necessary and sufficient tests will decrease the

quality risk even if sometimes they did not find more faults. We define risk compensation ability to evaluate the ability of compensating the omitted necessary tests by actual CI testing. It is defined as Equation 3.

$$Riskcomp = \frac{|St| - |(St@F \cup St@C) \cap Ct|}{|St@F \cup St@C|} \quad (3)$$

Here, St is the test set of the measured approach, it can be $St@F$ or $St@C$ (for $Riskcomp$ of FEST or ClassSRTS). $St@F \cup St@C$ means the union set of tests selected by FEST and ClassSRTS respectively. $(St@F \cup St@C) \cap Ct$ is the test set running in actual CI testing and also selected by FEST or baseline. The positive $Riskcomp$ means that the measured approach can compensate risk for actual CI testing, the higher value of $Riskcomp$ the better. In contrast, the negative $Riskcomp$ means it can not compensate risk. When $|St@F \cup St@C|$ is 0, we simply treat $Riskcomp$ as 0.

IV. EXPERIMENT RESULTS AND ANALYSIS

Based on the number of faults detected by actual CI testing and FEST, we divide the experiment project versions (261) into four categories. The following subsections will present their results and analysis respectively.

- Category *Dual_F* (9%:23): both actual CI testing and FEST detected faults.
- Category *FEST_F* (20%:51): actual CI testing did NOT detect faults, while FEST detected faults.
- Category *Dual_NF* (70%:184): both actual CI testing and FEST did NOT detect faults.
- Category *CI_F* (1%:3): actual CI testing detected faults, while FEST did NOT detect faults.

A. Category *Dual_F*

In Figure 2, the size of bubbles refers to the number of tests. We order versions by reduced test size between FEST and actual CI testing, then assign the ID sequentially.

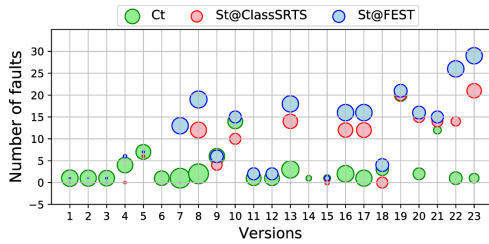


Fig. 2. Number of selected tests and number of detected faults for Dual_F

We can easily see that blue bubbles are either higher or inside other bubbles. It means that FEST can find more or equal number of faults than ClassSRTS or actual CI testing. Particularly, in some versions, the difference is quite large.

1) *Fault detection efficiency*: In Figure 3, FEST can not only cover all faults detected by actual CI testing and ClassSRTS, but also find new faults in 65% (15/23) and 83% (19/23) versions respectively. While ClassSRTS can cover the faults detected by actual CI testing only in 43% (10/23) versions, and find new faults in 48% (11/23) versions.

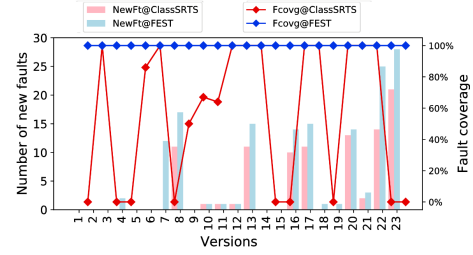


Fig. 3. Fault coverage and detection enhancement compared with actual CI testing for Dual_F

2) *Cost effectiveness*: In Figure 2, we observe that FEST selects slightly more tests than ClassSRTS in some versions. We further analyze the cost effectiveness of FEST.

Test scale benefits: From Figure 4, we can see that FEST shows better *Testscal* than actual CI testing in 96% (22/23) versions. Only in one version (v21), both FEST and ClassSRTS show slightly lower *Testscal* than actual CI testing. Compared with ClassSRTS, FEST shows better or equal *Testscal* in 87% (20/23) versions; in other three versions (v2, v19, v20), FEST shows lower *Testscal*. For these 4 versions where FEST shows lower *Testscal*, it has higher *Riskcomp* (see the next paragraph), which indicates our proposed approach can mitigate the risk of omitting necessary tests.

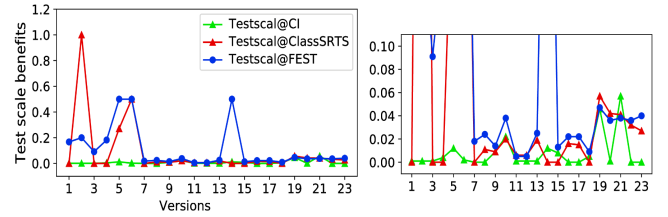


Fig. 4. Test scale benefits for Dual_F (the enlarged figure on the right shows the details)

Risk compensation ability: In Figure 5, compared with actual CI testing, FEST has positive *Riskcomp* in all versions, while ClassSRTS shows negative *Riskcomp* in 35% (8/23) versions. It means that in these 35% versions, ClassSRTS omits some necessary tests, which might lead to omit faults.

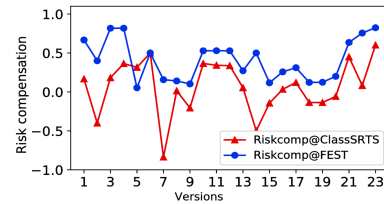


Fig. 5. Risk compensation ability for Dual_F

Compared with ClassSRTS, FEST shows higher *Riskcomp* in 96% (22/23) versions. It indicates that FEST can compensate more necessary tests than ClassSRTS, even though FEST selects more test in some cases. Only in 1 versions (v5), FEST shows lower *Riskcomp* than ClassSRTS. This is because when computing *Riskcomp*, we treat the union set of tests selected by FEST and ClassSRTS as ground truth of necessary tests. However, ClassSRTS has selected some unnecessary tests. In detail, it treats the selected tests as the difference between the set of all tests and the set of non-affected tests (tests not affected by changed code). However, the set of all tests could include some unnecessary classes, e.g., the basic classes which

do not have test cases. Hence, the lower $Riskcomp@FEST$ does not indicate the low effectiveness of our approach.

B. Category FEST_F

Similar to category Dual_F, in Figure 6, the size of bubbles refers to the number of tests.

1) *Fault detection efficiency*: In Figure 6, all green bubbles lie on X-axis due to $|C_{ft}| = 0$. It indicates that compared with actual CI testing, the fault coverage of both FEST and ClassSRTS are all 100%. More than that, FEST can cover all faults detected by ClassSRTS and detect more new faults in 57% versions.

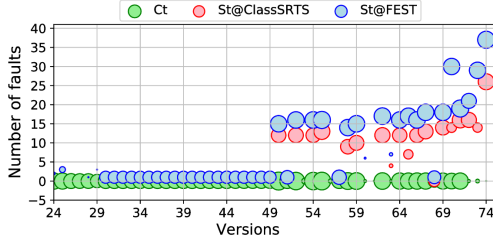


Fig. 6. Number of selected tests and number of detected faults for FEST_F

2) *Cost effectiveness*: In Figure 6, test size of FEST is smaller than actual CI testing, and similar or slightly larger than ClassSRTS.

Test scale benefits: In this category, $Testscal@CI = 0$ in all versions. From Figure 7, we can observe that FEST shows better $Testscal$ than actual CI testing, and better or equal $Testscal$ than ClassSRTS in all versions. Especially, in 8 versions, $Testscal$ of FEST is more than three times as large as that of ClassSRTS.

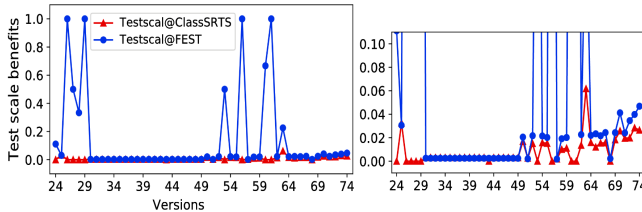


Fig. 7. Test scale benefits for FEST_F (the enlarged figure on the right shows the details)

Risk compensation ability: In Figure 8, FEST has positive $Riskcomp$ in 90% (46/51) versions and none negative $Riskcomp$, while ClassSRTS shows negative $Riskcomp$ in 24% versions. Meanwhile, compared with ClassSRTS, FEST has better or equal $Riskcomp$ in 96% (49/51) and 2% (1/51) versions respectively. The reason of lower $Riskcomp$ in other 1 version (v63) is similar with the discussion in Section IV-A2.

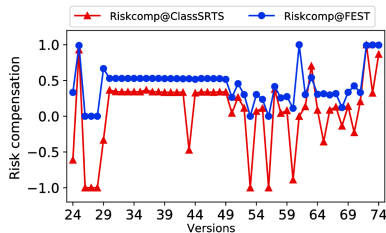


Fig. 8. Risk compensation ability for FEST_F

C. Category Dual_NF

In all versions of this category, neither actual CI testing nor FEST or ClassSRTS detected faults (i.e., $Fcovg$, $Fenhm$ and $Testscal$ are 0). Therefore, we only discuss risk compensation ability for this category.

Risk compensation ability: In Figure 9, FEST has positive $Riskcomp$ in 49% versions and none negative $Riskcomp$, while ClassSRTS shows positive and negative $Riskcomp$ in 28% and 47% versions respectively. Compared with ClassSRTS, FEST presents higher or equal $Riskcomp$ than ClassSRTS in 91% (118/261) versions, among which $Riskcomp$ of FEST is twice as large as that of ClassSRTS in 109 versions. The reason of lower $Riskcomp$ in 9% versions is similar with the discussion in Section IV-A2.

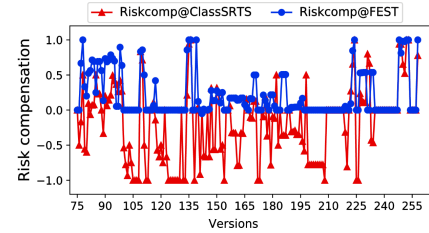


Fig. 9. Risk compensation ability for Dual_NF

D. Category CI_F

1) *fault detection efficiency*: In all three versions (i.e., v259, v260 and v261) of this category, actual CI testing detected a few faults while ran a large size of tests. Both FEST and ClassSRTS selected very small test sets while detected no faults. We further examine the detail of these 3 versions.

In the 3 versions, the faults which are not covered by FEST do not have any dependencies with the changed and affected code, and have no failed test history in recent 10 versions. In other words, these faults were committed in earlier versions, and should be detected earlier. Hence, detecting these faults is not the responsibility of current CI testing. This indicates that the bad performance does not due to the drawback of FEST.

2) *Cost-effectiveness*: In this category, both FEST and ClassSRTS did not detect any faults, $Testscal$ of both FEST and ClassSRTS are zero, lower than actual CI testing.

Risk compensation ability: Regarding risk compensation ability, both FEST and ClassSRTS positively compensate risk in all versions, but FEST can get much higher $Riskcomp$ for all of them even though it did not find any faults. It is because actual CI testing ran some redundant tests while omitting necessary tests which results in a high risk of omitting faults. Meanwhile, FEST presents better or equal $Riskcomp$ than ClassSRTS in all versions.

Summary: Based on the detailed results and analysis of the 4 categories, we obtain the summary of FEST across all categories: (1) FEST can cover all faults detected by actual CI testing and baseline, and find new faults in 25% and 18% versions respectively; (2) FEST shows better or equal $Testscal$ than actual CI testing (in 98% versions) and ClassSRTS (in 99% versions); can compensate risk of omitting necessary tests for actual CI testing (in 62% versions) and baseline (in 73% versions).

The **internal** validity of our study arises from the implementation of baseline and FEST. We implemented baseline by strictly following the steps described in the original paper [13]. For both baseline and FEST approaches, we have employed 213 test cases to test their functionality. For the suspicious results, we have manually checked the code, and found they do not due to the defect in code.

The **external** validity concerns about our experimental dataset. We can not guarantee that our results can be fully generalize to other projects. However, the size of the dataset (18 projects with 261 versions) and the diversity of domains relatively reduce this risk.

VI. RELATED WORK

CI test selection based on dynamic dependency: Gligoric et al. [7] and Vasic et al. [19] implemented a test selection approach based on dynamic dependencies at the file level for *Java* and *.NET* respectively, and analyzed the strengths and drawbacks of file-level and module-level test selections. Because of that, Zhang [15] proposed a hybrid test selection technique that combined file-level and method-level analysis. Furthermore, Celik et al. [8] designed a file-level test selection across *JVM* boundaries, which can find more dependencies and improve precision of selection.

CI test selection based on static dependency: Soetens et al. [25] and Parsai et al. [4] proposed an approach to select tests on method-level static dependencies and improved it to deal with invocation in polymorphism. Legunsen et al. [5], [13] implemented class-level and method-level test selection techniques based on static dependencies. But the class-level approach would still omit some tests because it can not obtain all dependencies (e.g., the dependencies in reflection and cast operation), while the method-level approach shows worse performance in fault detection and time cost.

Generally speaking, the drawbacks with existing dynamic selection approaches are long running tests, non-determinism, and real-time constraints; while the drawbacks of existing static selection approaches lie in omitting some tests or selecting some unnecessary tests [4]–[6], [13], [20]. Our approach belongs to static test selection which improves existing approaches by resolving full dependency relations and selecting a test set towards fully covering all changed and affected code.

VII. CONCLUSION

In this paper, we propose FEST to select a proper test subset towards full coverage of all changed and affected code so as to reduce the cost of CI testing. Evaluations are conducted on 18 projects with 261 CI versions from Eclipse and Apache communities. Results show that FEST can outperform actual CI testing and the state-of-the-art ClassSRTS in terms of fault detection efficiency and cost-effectiveness in most cases.

VIII. ACKNOWLEDGMENTS

We sincerely thank Yun Yang for his contribution to the paper. This work is supported by National Natural Science Foundation of China under Grant No.61602450, No.61432001.

- [1] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, “Quality and productivity outcomes relating to continuous integration in github,” in *FSE’15*, pp. 805–816.
- [2] S. Elbaum, G. Rothermel, and J. Penix, “Techniques for improving regression testing in continuous integration development environments,” in *FSE’14*, pp. 235–245.
- [3] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, “Usage, costs, and benefits of continuous integration in open-source projects,” in *ASE’16*, pp. 426–437.
- [4] A. Parsai, Q. D. Soetens, A. Murgia, and S. Demeyer, “Considering polymorphism in change-based test suite reduction,” in *Agile’14*, pp. 166–181.
- [5] O. Legunsen, F. Hariri, A. Shi, Y. Lu, L. Zhang, and D. Marinov, “An extensive study of static regression test selection in modern software evolution,” in *FSE’16*, pp. 583–594.
- [6] V. Blondeau, A. Etien, N. Anquetil, S. Cresson, P. Croisy, and S. Ducasse, “Test case selection in industry: An analysis of issues related to static approaches,” *SQJ’16*, pp. 1–35.
- [7] M. Gligoric, L. Eloussi, and D. Marinov, “Practical regression test selection with dynamic file dependencies,” in *ISSTA’15*, pp. 211–222.
- [8] A. Celik, M. Vasic, A. Milicevic, and M. Gligoric, “Regression test selection across JVM boundaries,” in *FSE’17*, pp. 809–820.
- [9] K. Herzig, M. Greiler, J. Czerwonka, and B. Murphy, “The art of testing less without sacrificing quality,” in *ICSE’15*, pp. 483–493.
- [10] A. Memon, Z. Gao, B. Nguyen, S. Dhandia, E. Nickell, R. Siemborski, and J. Micco, “Taming google-scale continuous testing,” in *ICSE’17*, pp. 233–242.
- [11] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, “Reinforcement learning for automatic test case prioritization and selection in continuous integration,” in *ISSTA’17*, pp. 12–22.
- [12] D. Mondal, H. Hemmati, and S. Durocher, “Exploring test suite diversification and code coverage in multi-objective test case selection,” in *ICST’15*, pp. 1–10.
- [13] O. Legunsen, A. Shi, and D. Marinov, “Starts: Static regression test selection,” in *ASE’17*, pp. 949–954.
- [14] A. Shi, T. Yung, A. Gyori, and D. Marinov, “Comparing and combining test-suite reduction and regression test selection,” in *FSE’15*, pp. 237–247.
- [15] L. Zhang, “Hybrid regression test selection,” in *ICSE’18*, pp. 199–209.
- [16] K. Wang, C. G. Zhu, A. Celik, J. Kim, D. Batory, and M. Gligoric, “Towards refactoring-aware regression test selection,” in *ICSE’18*, pp. 233–244.
- [17] G. Wikstrand, R. Feldt, J. K. Gorantla, and C. Zhe, W. and White, “Dynamic regression test selection based on a file cache: An industrial evaluation,” in *ICST’09*, pp. 299–302.
- [18] E. D. Ekelund and E. Engström, “Efficient regression testing based on test history: An industrial evaluation,” in *ICSME’15*, pp. 449–457.
- [19] M. Vasic, Z. Parvez, A. Milicevic, and M. Gligoric, “File-level vs. module-level regression test selection for .Net,” in *FSE’17*, pp. 848–853.
- [20] Q. D. Soetens, S. Demeyer, A. Zaidman, and J. Pérez, “Change-based test selection: An empirical evaluation,” *ESE’16*, vol. 21, no. 5, pp. 1990–2032.
- [21] A. Orso, N. Shi, and M. J. Harrold, “Scaling regression testing to large software systems,” *Acm Sigsoft Software Engineering Notes*, vol. 29, no. 6, pp. 241–251, 2004.
- [22] B. Meyer, *Object-Oriented Software Construction*. Prentice Hall, New York, N.Y., second edition, 1997.
- [23] S. Yoo, R. Nilsson, and M. Harman, “Faster fault finding at google using multi objective regression test optimization,” in *ESEC/FSE’11*, pp. 1–4.
- [24] A. Beszédés, T. Gergely, L. Schrettnner, J. Jász, L. Langó, and T. Gyimóthy, “Code coverage-based regression test selection and prioritization in webkit,” in *ICSM’12*, pp. 46–55.
- [25] Q. D. Soetens, S. Demeyer, and A. Zaidman, “Change-based test selection in the presence of developer tests,” in *CSMR’13*, pp. 101–110.
- [26] S. Mirarab, S. Akhlaghi, and L. Tahvildari, “Size-constrained regression test case selection using multicriteria optimization,” *TSE’12*, pp. 936–956.
- [27] E. Engström, P. Runeson, and A. Ljung, “Improving regression testing transparency and efficiency with history-based prioritization - an industrial case study,” in *ICST’11*, pp. 367–376.

Amplifying Tests for Cross-Platform Apps through Test Patterns

Thiago Botti de Assis, André Augusto Menegassi, and Andre Takeshi Endo

Department of Computing

Federal University of Technology – Parana (UTFPR)

Cornélio Procopio, Brazil

thiagobotti@gmail.com, andremenegassi@hotmail.com, andreendo@utfpr.edu.br

Abstract—Cross-platform app development has achieved meaningful results in practice with frameworks like React Native, Xamarin, and Apache Cordova. Unlike native apps, such frameworks support the development of a mobile app that can run in different platforms. Nevertheless, the literature lacks techniques to test cross-platform apps since most of the existing works focus on native Android apps. A promising strategy for native apps is to amplify test suites so that the specific characteristics of mobile apps can be tested. This paper aims to investigate the test amplification of cross-platform apps. To do so, we apply four test patterns that verify well-known characteristics of mobile computing and amplify existing test suites. The proposed approach has been implemented in a tool capable of generating Appium test scripts and was evaluated with nine cross-platform apps. The amplified test suites exercise new scenarios, uncovering 23 unique bugs in eight out of nine apps.

I. INTRODUCTION

Smartphones, tablets, e-readers, and wearables have dominated the consumption of digital information and data traffic; the number of these devices has grown exponentially over the years [1]. The software running in such devices, so-called *mobile apps* (or just *apps* for short), radically changed the lifestyle of billions of people around the world, being used for several hours every day and helping users to perform a wide variety of activities [2]. To meet the ample range of users, millions of apps have been developed and are available for download [3]. Currently, the dominant operating systems (OSs) for mobile devices are Google’s Android and Apple’s iOS; they provide their own software development kits, characterizing the single-platform native development.

In order to reach more end users, the software industry has shown an upward trend in adopting cross-platform mobile app development. In such a paradigm, a unique code base is used to deliver the same app in different platforms (e.g., Android, iOS, Windows) and their versions [4], [5]. This is aided by several open source and commercial cross-platform development frameworks and tools; the most notable examples are React Native, Xamarin, and Apache Cordova. Developed by Facebook, React Native leverages the well-known web library React to generate mobile apps with native user interfaces (UIs) and coded in Javascript [6]. Acquired by Microsoft in 2016, Xamarin offers C# or F# as a common programming language to develop apps for the three main platforms: Android, iOS and

Windows. According to Microsoft, 75% of the code is shared among the platforms on average; this figure can be close to 100% if the UI toolkit Xamarin.Forms is employed [7]. Finally, Apache Cordova fosters the development of hybrid apps using standard web technologies like HTML5, CSS3, and Javascript [8]. As such, it is popular among web developers and has supported other mobile frameworks like Ionic and PhoneGap [9], [10].

Apart from this trend, research efforts in engineering high-quality mobile apps have focused on native development for Android [11], [12]. The demand for quality in mobile apps has grown along with their spread; the users want the apps to be reliable, robust, and efficient. As a consequence, mobile software engineers must adopt adequate quality assurance techniques [2]. Software testing is the primary activity to reduce risks and minimize the presence of bugs in the software. In a nutshell, testing involves the process of running a software system with the purpose of uncovering bugs [13].

As mobile apps possess specific features to be checked, *Test Amplification* [14] has been investigated in literature to go beyond functional test cases. Test amplification can be defined as the extension of existing test cases to verify other properties of the software under test. This is a promising strategy for mobile apps that are susceptible to bugs related to device rotation, interruptions, loss of resources, unique UI events/components, and so on. The verification of such specific features can be described in a well-defined set of generic actions and expected behavior called *test pattern* [15], [19]. The amplification by test patterns has been explored in native Android apps and results have shown effective bug detection [15], [16], [18], [19]. However, little is known on how test amplification performs in mobile apps developed with cross-platform frameworks.

This paper introduces an approach for test amplification of cross-platform apps. In particular, we describe four test patterns that aim to verify well-known characteristics of mobile computing and amplify test cases. The proposed approach has been implemented in a tool, called x-PATeSCO, capable of producing test scripts for cross-platform apps. We evaluated the approach and its supporting tool with nine real-world apps, developed with frameworks React Native, Xamarin, and Cordova.

This paper is organized as follows: Section II presents the

background and related work. Section III brings the test amplification approach proposed for cross-platform apps. Section IV presents the evaluation setup and the results are analyzed in Section V. Finally, Section VI makes the concluding remarks and sketches future work.

II. BACKGROUND AND RELATED WORK

Mobile computing brings a diverse set of features that may cause failures in the software developed for such platform. Holl and Elberzhager [22] propose a failure classification based on a literature review and a study with developers and testers of mobile apps. They list events that might make apps to fail, namely, temporarily disconnected network, network change (3G - WiFi), GPS temporarily disconnected, weak battery, globalization (localization, language, date or time differences between clients and servers), UI changes due to orientation, screen changes, OS-specific native functionality, interruptions via incoming calls or text messages, unexpected termination of an app process, and permissions. Similarly, Linares-Vázquez et al. [21] manually searched for bugs in Android apps related to specific features like GPS failures, loss of access to network data, slow data reception, and device rotation. The authors used this bug set to elaborate Android-specific mutation operators and developed a mutation testing tool for Android apps.

These well-known features have also been used to amplify existing test cases in a systematic way. Zaeem et al. [16] also performed a study with bugs from the most popular open source app projects, identifying feature interactions that might cause errors. The authors proposed an approach to verify automatically the feature interactions and a tool was developed to exhaustively test them. Morgado and Paiva [15] introduce a catalog of behavioral principles of the Android UI elements. Such behavioral principles are described as test patterns related to UI menu elements (drawer style) and device rotation. They also propose patterns related to features like GPS and 3G connection. Adamsen et al. [18] explore the injection of adverse conditions in existing test suites. The adverse conditions are related to common usage of apps like pause-stop-resume, rotation, as well as missing or changed resources such as loss of GPS accuracy, change of mobile network, and transition between Internet networks (3G-WiFi-3G). Their results show that the approach is effective in finding critical flaws and the additional cost in the test time is low, compared to the number of bugs found.

While test amplification and test patterns have been investigated [15], [16], [18], the state of the art in mobile app testing is focused on native Android apps [11], [12]. On the other hand, cross-platform app development has gained momentum by winning a meaningful market share, and being supported by big players of the software industry like Facebook, Microsoft, and Apache Software Foundation. In this context, one might wonder if the testing techniques developed for native apps are seamlessly applicable to cross-platform apps. In this paper, we inquire into test amplification for cross-platform apps; to our best knowledge, this is the first study on the topic.

III. TEST AMPLIFICATION APPROACH

This section describes the approach we define to test amplification of cross-platform apps; Figure 1 shows an overview of the approach. There is a component called Test Amplification Generator that receives as input a set of system-level test cases TC_1, TC_2, \dots, TC_n and a one or more test patterns. Then, it produces as output a combination of test cases and test patterns, the *amplified test script*. Such scripts can then be executed in the *app under test* (AUT).

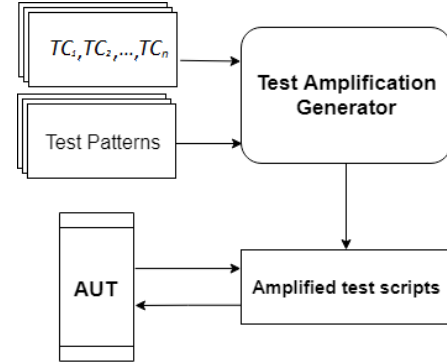


Fig. 1. Overview of the proposed approach.

The following sections explain how we implemented the approach. Section III-A details how a test case and a test pattern are combined to amplify the tests. In particular, we leverage four test patterns that verify well-known characteristics of mobile apps. Section III-B focuses on the Test Amplification Generator, describing how we made the approach applicable to cross-platform apps.

A. Test Patterns

We herein describe the four test patterns adopted in this work. All test patterns assume that a test case is provided as a sequence of system-level events, mostly UI actions like clicking a button, selecting an item, or filling a text field. The test pattern contains not only a pre-defined sequence of specific events that should be executed, but also a mechanism to identify a potential bug (i.e., a test oracle). To simplify the presented algorithms, we do not treat exceptions that can be thrown when a given event is executed in the app (e.g., a crash). Nevertheless, our tool deals with it by capturing the exception and recording the bug information in a log (further details in Section III-B). The literature on Android app testing reports different strategies to amplify test suites [4], [16], [18], [21]; this study adopts four test patterns, well-known in literature and that we found relevant to cross-platform apps.

1) *Lost Connection (LC)*: This pattern aims to verify how the AUT behaves when the Internet connection is lost [15], [21], [22]. As mobile devices are susceptible to unstable networks, the AUT is supposed to be aware and react when a connection is lost. By reacting, we assume that some feedback is provided to the end user, like an error message or a warning [17], [20].

Algorithm 1 describes how a test case is amplified to verify a lost connection. It receives as input a test case as an array of events, and a map of states recorded during the original execution of the test case. First, the procedure turns the connection off and starts the app (Lines 4-5). In each iteration, it checks if the current state is different from the original state (Line 8); i.e., the connection lost had an impact on the UI. If the current state has no sign of feedback for the user (like an error message) (Line 9), we record the scenario as a bug (Line 10). To resume the verification in the following events of the test case, we turn the connection on (Line 12), execute the events normally until the current state (Line 13). The connection is then turned off (Line 14) and the iteration continues to the next event (Line 16).

Algorithm 1 Lost Connection Test Pattern

```

1: procedure LOSTCONNECTION(allEvents[], origStates[])
2:   input allEvents[] - Test case as an array of events
3:   input origStates[] - Map of states collected from the original execution
4:   connection( OFF );
5:   startApp();
6:   for each event ∈ allEvents do
7:     currentState = getActualState();
8:     if currentState != origStates[event] then //changed GUI
9:       if ! existLostConnectionMsg(currentState) then //no feedback
10:        RecordBugInfo();
11:       end if
12:       connection( ON );
13:       bringAppTo( currentState );
14:       connection( OFF );
15:     end if
16:     executeEvent(event);
17:   end for
18: end procedure

```

2) *Back Event (BE)*: This pattern aims to verify how the AUT behaves when the Back event is triggered during a test case [16], [22]. We assume that when running, the Back event must go to a previous screen of the application [17], [20].

Algorithm 2 describes how the test case is amplified to verify the execution of the Back event. It receives as input a test case as an array of events. Initially, the procedure starts the application (Line 3). In each iteration, we execute the corresponding event, then it is checked if the state after the event execution is equal to the state before execution (Line 8). The procedure executes the Back event, and validates whether the application has returned to the previous state (Lines 9-11). If the states are different after running the Back event, we record this scenario as a bug (Line 12). If the states are equal, the procedure follows the verification for the next event.

3) *Side Drawer Menu (SDM)*: This pattern aims to verify how AUT behaves when having a side menu [16], [22]. The menu opens when the user slides from the screen to the center or clicks in a menu icon or label [15]. We validate if all the menu items take the app to a different screen.

Algorithm 3 describes how the test case is amplified to validate the application menus. It receives as input a test case as an array of events. Initially, the procedure starts the application (Line 3). For each iteration, we check whether the application has a menu (Lines 6-23). If there is a menu in

Algorithm 2 Back Event Test Pattern

```

1: procedure BACKEVENT(allEvents[])
2:   input allEvents[] - Test case as an array of events
3:   startApp();
4:   for each event ∈ allEvents do
5:     beforeState = getActualState();
6:     executeEvent(event);
7:     afterState = getActualState();
8:     if beforeState != afterState then //there is a screen transition
9:       ExecuteBack();
10:      afterBackState = getActualState();
11:      if beforeState != afterBackState then //bug detected
12:        RecordBugInfo();
13:      end if
14:      bringAppTo( afterState );
15:    end if
16:  end for
17: end procedure

```

the AUT, the procedure checks the menu items empirically by clicking on them and checking if the application changes the screen after entering a menu item (Lines 28-33). If the application does not change screen, we record this scenario as a bug (Line 31). If no menu item is found, the procedure proceeds to the next iteration (Line 35).

Algorithm 3 Side Drawer Menu Test Pattern

```

1: procedure SIDEDRAWERMENU(allEvents[])
2:   input allEvents[] - Test case as an array of events
3:   startApp();
4:   for each event ∈ allEvents do
5:     beforeMenuState = getActualState();
6:     haveMenu = false;
7:
8:     //Try do detect menu by horizontal swipe
9:     initialState = beforeMenuState;
10:    Swipe();
11:    finalState = getActualState();
12:    if initialState != finalState then
13:      haveMenu = true;
14:    else
15:      //Try do detect menu by menu button
16:      initialState = getActualState();
17:      LocateAndClickMenuButton();
18:      finalState = getActualState();
19:
20:      if initialState != finalState then
21:        haveMenu = true;
22:      end if
23:    end if
24:
25:    if haveMenu then
26:      menuState = getActualState();
27:      for each item ∈ getMenuItems() do
28:        tap( item );
29:        afterTapItemState = getActualState();
30:        if afterTapItemState ∈ {beforeMenuState, menuState} then
31:          RecordBugInfo();
32:        end if
33:        bringAppTo(menuState);
34:      end for
35:      bringAppTo(beforeMenuState);
36:    end if
37:    executeEvent(event);
38:  end for
39: end procedure

```

4) *Don't Change State (DCS)*: This pattern aims to verify how the AUT behaves when external events triggered by the user occur during the test case [15], [16], [18]. As mobile devices are susceptible to these actions, the AUT is supposed

to be aware and prevent failures because of external events. We assume that after the execution of external events, the app should return to the state it was before [17], [20].

Algorithm 4 describes how a test case is amplified to verify the external events performed by users. It receives as input a test case as an array of events, and the type of the external action to be executed. The types currently supported are Rotate, Pause-Stop-Resume, Zoom, Swipe, and Scroll. First, the procedure starts the application (Line 4). For each iteration, it records the state before and after the external event (Lines 7-9). If the states are different (the AUT did not return to the same screen) (Line 10), we record the scenario as a bug (Line 11). The procedure then continues to the next event (Line 12).

Algorithm 4 Don't Change State Test Pattern

```

1: procedure DONTCHANGESTATE(allEvents[], type)
2:   input allEvents[] - Test case as an array of events
3:   input type ∈ {Rotate, PauseStopResume, Zoom, Scroll, Swipe}
4:   startApp();
5:   for each event ∈ allEvents do
6:     executeEvent(event);
7:     beforeState = getActualState();
8:     execute( type ); //execute one of the 5 types
9:     afterState = getActualState();
10:    if beforeState != afterState then //didn't come back to the same screen
11:      RecordBugInfo();
12:      bringAppTo( beforeState );
13:    end if
14:  end for
15: end procedure

```

B. Test Amplification Generator

We developed a tool named x-PATeSCO (*cross-Platform App Test Script reCOder*) to support cross-platform mobile app testing. The tool is built on top of Appium [23], an open source framework to automate UI tests in native, Web and hybrid apps. Moreover, Appium is cross-platform and makes it possible to automate tests for iOS and Android platforms, using a Selenium WebDriver API. x-PATeSCO is able to read the elements of an app's UI and assist the tester in selecting these elements to design automated test cases.

The Test Amplification Generator component has been implemented within x-PATeSCO. The four test patterns aforementioned are embedded into the tool as code templates, that are integrated with the recorded test cases. Such integration is realized when the test scripts are generated; currently, x-PATeSCO produces scripts coded in C#. These scripts are capable of executing the four test patterns in the original test case automatically, therefore no manual effort is required to apply test amplification in cross-platform apps.

IV. EVALUATION SETUP

To evaluate the proposed approach, we set out the following research questions (RQs):

- **RQ₁**. Can test amplification detect bugs in cross-platform apps?
- **RQ₂**. What is the impact of test amplification on test execution?

- **RQ₃**. Are the results replicable in different settings (e.g., OS version)?

To answer **RQ₁**, we record the number of unique bugs detected as a consequence of the test amplification. As our approach may show false alarms, the number of tests presenting failure (failures) and false positives are also collected. So, for each failure, we try to reproduce it by manually performing the test steps. For a successful reproduction, we have a bug, otherwise a false positive. As for **RQ₂**, we measure the overhead of the amplified tests in test execution time. Given that T_{orig} is the runtime of the original test cases and T_{amp} is the runtime of the amplified tests, the overhead is calculated by T_{amp}/T_{orig} . Finally, **RQ₃** analyzes the results collected in **RQ₁** from the perspective of different configurations. In this study, we only considered different versions of the OS Android.

Table I lists the nine apps evaluated; for each app, it shows the project size in number of lines of code (LOC), the cross-platform app development framework, the number of test cases (#TCs) and their events (#Ev.). Such cross-platform apps' size ranges from around 400 LOC to up to 178 KLOC, and are implemented in different cross-platform development frameworks (namely, Apache Cordova, React Native, and Xamarin). Two apps are industrial (namely, MemesPlay and Bargains) and have been provided by partner IT companies. The other seven are open source projects obtained from GitHub. Each app has two or three test cases (with nine to 20 events each), designed by independent participants.

TABLE I
APPS UNDER TEST.

App	Size-LOC	Framework	#TCs (#Ev.)
Fresh-Food-Finder	13824	Cordova	3 (20)
OrderApp	71565	Cordova	3 (16)
MemesPlay	5484	Cordova	3 (12)
Agenda	1038	Cordova	3 (18)
ToDoListCordova	9304	Cordova	3 (10)
MovieApp	2088	React Native	3 (10)
ToDoList	405	React Native	3 (13)
Tasky	654	Xamarin	3 (9)
Bargains	178266	Xamarin	2 (10)

x-PATeSCO was used to support the experimental evaluation; it connects to Appium which, in turn, connects to mobile devices to run the test cases and log pieces of information to answer **RQ₁**, **RQ₂**, and **RQ₃**. The apps were installed in two devices, one with Android 6.0 and another with Android 7.0, and all tests were executed in both devices. Besides the mobile devices, the test projects were run from a computer with an Intel Core i5 dual-core (2.5 GHz) processor and 8 GB of RAM without any further processing or communication load in order to avoid CPU and memory saturation. An Appium server was also installed in this machine.

To foster future replication and overcome possible threats to this study, we make the tool and the experiment objects available as an open source experimental package in <https://goo.gl/7CF4oZ>

V. ANALYSIS OF RESULTS

a) *Bug detection*: Table II shows the number of bugs found, failing test cases (Fail), and false positives (FP); rows identify the apps and OS version (OS V.), while columns group the test patterns: *Back Event (BE)*, *Lost Connection (LC)*, *Side Drawer Menu (SDM)*, and *Don't Change State (DCS)*. Column *DCS* takes into account the results of Rotate, PauseStopResume, Zoom, Scroll, and Swipe. Symbol '-' indicates that the test pattern is not applicable in a given app. For instance, Agenda does not use Internet connection, so *LC* is not applicable. In this part, we focus our analysis on the OS version that had more bugs revealed (namely, Android 6.0).

As for *BE*, this pattern revealed seven bugs in seven different apps. We did not observe false positives since all failures were caused by the identified bugs. In general, this pattern finds a faulty scenario in which the app does not return to a previous screen using the back button. Instead, the app terminates, goes to the background, or returns to a different screen. Concerning *LC*, this pattern revealed five bugs in four out of five apps in which is applicable. It shows a smaller number of failures and no false positives. This is likely due to the low number of events that depend on an Internet connection to properly execute. For instance, OrderApp does not give a feedback on a lost message due to a failing Internet connection. MovieApp presents the same bug, and in other scenario of connection lost the app crashes.

There is no bug, failure or false positive for *SDM*. While four apps have menus, all menu items are accessible and navigate to different screens correctly. We surmise that menus are built with well-tested UI components and developers pay special attention to them due to their relevancy in the app. As a consequence, menus are more tested and less susceptible to bugs. The *DCS* test pattern uncovered 11 bugs in eight apps. Notice that this pattern showed a high number of failures and false positives. This occurred due to issues related to the Appium framework. Actions like Zoom, Scroll and Swipe caused exceptions during the automated test execution, but we failed to reproduce these scenarios manually. As for the bugs, the OrderApp, TodoList, Bargains, and Memesplay apps restarted after a Pause-Stop-Resume action. Agenda, Fresh-Food-Finder, MovieApp, TodoList, and Bargains slowed down when performing the zoom action.

The amplified tests detected 23 unique bugs in eight out of the nine cross-platform apps. Among the four test patterns, only *SDM* did not reveal a bug. A reasonable number of false positives has been observed, except for *DCS*. We believe that new Appium versions and adjustments in x-PATeSCO can reduce the presence of false alarms in future. The results give evidence that supports a positive answer for **RQ₁**.

b) *Overhead in test execution*: Table III shows the overhead of the amplified tests in test execution time. The overhead of the test patterns shows some variation among the apps. For

instance, the *BE*'s overhead ranges from 0.71 to 4.5 times in comparison with the original test suite. On average, *BE* also poses the highest overhead (2.97x), while *SDM* shows the lowest average (2.06x). *DCS* and *LC* had intermediate averages, 2.42x and 2.68x, respectively.

TABLE III
OVERHEAD IN THE TEST EXECUTION TIME.

App	OS V.	BE	LC	SDM	DCS
Fresh-Food-Finder	6.0	4.50	1.70	1.71	2.59
	7.0	4.55	1.72	1.70	2.69
OrderApp	6.0	4.25	3.37	-	3.91
	7.0	3.36	2.66	-	3.09
Memesplay	6.0	4.25	5.56	3.09	4.47
	7.0	2.40	3.58	1.96	2.70
Agenda	6.0	3.44	-	1.82	2.07
	7.0	3.42	-	1.64	2.29
TodoListCordova	6.0	3.71	-	-	2.62
	7.0	4.17	-	-	2.87
MovieApp	6.0	2.20	2.54	3.10	2.04
	7.0	0.71	0.82	1.02	0.65
TodoList	6.0	1.43	-	-	2.40
	7.0	1.39	-	-	1.99
Tasky	6.0	1.97	-	-	2.38
	7.0	2.00	-	-	2.43
Bargains	6.0	2.22	3.04	-	1.77
	7.0	2.50	2.31	-	1.99
Max		4.50	5.56	3.10	4.47
Min		0.71	0.82	1.02	0.65
Avg		2.97	2.68	2.06	2.42

The results show that amplified tests execute from 0.65 to up to 5.56 times the runtime of the original test suite. Answering **RQ₂**, the impact can be reasonable depending on the app and test pattern, though the (low) cost of CPU time might be compensated by the bug detection capabilities of the proposed approach.

c) *Different settings*: Due to space limitations, we analyze the results using two settings: Android 6.0 and Android 7.0. As we used two devices with different hardware configurations, variations in the runtime were expected and natural (see Table III). So, we focus on the results summarized in Table II. Notice that there is no discrepancy between the OS versions for test patterns *BE*, *LC*, and *SDM*. So, the 12 bugs detected by them were detected in both devices. From the 23 bugs found in Android 6.0, 11 were not observed in Android 7.0. Fewer failures were also noticed in Android 7.0, yet the number of false positives was slightly higher.

As all false positives were caused by Appium's issues, newer versions of Android seem to be more robust for automated tests. Surprisingly, all bugs found by *DCS* in Android 6.0 were not detected in Android 7.0. We believe that Android policies for background processes (app not in focus) may differ between the OS versions. This might cause a faulty behavior in a specific version.

The different results obtained in Android 6.0 and Android 7.0 give initial evidence that supports a negative answer to **RQ₃**. This motivates further investigation on different Android versions and other target OSs like iOS and Windows.

TABLE II
NUMBER OF BUGS, FAILURES (FAIL), AND FALSE POSITIVES (FP).

App	OS V.	BE			LC			SDM			DCS			Total		
		Bug	Fail	FP	Bug	Fail	FP	Bug	Fail	FP	Bug	Fail	FP	Bug	Fail	FP
Fresh-Food-Finder	6.0	1	10	0	0	0	0	0	0	0	1	31	15	2	41	15
	7.0	1	10	0	0	0	0	0	0	0	0	16	16	1	26	16
OrderApp	6.0	1	15	0	1	6	0	-	-	-	1	9	0	3	30	0
	7.0	1	15	0	1	6	0	-	-	-	0	9	9	2	30	9
Memesplay	6.0	1	8	0	1	6	0	0	0	0	1	23	21	3	37	21
	7.0	1	8	0	1	6	0	0	0	0	0	17	17	2	31	17
Agenda	6.0	1	8	0	-	-	-	0	0	0	1	40	38	2	48	38
	7.0	1	8	0	-	-	-	0	0	0	0	12	12	1	20	12
ToDoListCordova	6.0	1	3	0	-	-	-	-	-	-	1	11	10	2	14	10
	7.0	1	3	0	-	-	-	-	-	-	0	11	11	1	14	11
MovieApp	6.0	1	8	0	2	6	0	0	0	0	3	46	39	6	60	39
	7.0	1	8	0	2	6	0	0	0	0	0	46	46	3	60	46
ToDoList	6.0	0	0	0	-	-	-	-	-	-	1	18	13	1	18	13
	7.0	0	0	0	-	-	-	-	-	-	0	15	15	0	15	15
Tasky	6.0	0	0	0	-	-	-	-	-	-	0	2	2	0	2	2
	7.0	0	0	0	-	-	-	-	-	-	0	2	2	0	2	2
Bargains	6.0	1	3	0	1	5	0	-	-	-	2	21	5	4	29	5
	7.0	1	3	0	1	5	0	-	-	-	0	20	20	2	28	20
Total	6.0	7	55	0	5	23	0	0	0	0	11	201	136	23	279	143
	7.0	7	55	0	5	23	0	0	0	0	0	148	148	12	226	148

VI. CONCLUSION

This paper presented and evaluated an approach to generate amplified test scripts for cross-platform apps. To do so, we employed four test patterns that check well-known features of a mobile application: namely, *Lost Connection*, *Back Event*, *Side Drawer Menu*, and *Don't Change State*. We developed a supporting tool called x-PATeSCO and nine apps were used in the experiments. The results gave evidence that test amplification is capable of uncovering new bugs in cross-platform apps, though there exists a reasonable overhead in the test execution time. We also noticed variations between different versions of Android; this motivates further investigation on automated tests in multiple configurations of hardware, OSs, and so on.

ACKNOWLEDGMENT

Andre T. Endo is partially financially supported by CNPq/Brazil (grant number 420363/2018-1).

REFERENCES

- [1] Dzhangaryan, A. and Milenkovi, A. (2016) "Models for Evaluating Effective Throughputs for File Transfers in Mobile Computing", In: ICCCN 2016. doi: 10.1109/ICCCN.2016.7568547.
- [2] Amalfitano, D., Riccio, V., Paiva, A. C. R. and Fasolino, A. R. (2017) "Why does the orientation change mess up my Android application? From GUI failures to code faults", STVR journal (September), p. 1–27.
- [3] Rume, S. T. A. and Liu, D. (2013) "DroidTest: Testing Android Applications for Leakage of Private Information", In: ISC 2013.
- [4] Joorabchi, M. E., Ali, M. and Mesbah, A. (2015) "Detecting inconsistencies in multi-platform mobile apps", In ISSRE 2015, p. 450–460. doi: 10.1109/ISSRE.2015.7381838.
- [5] El-kassas, W. S., Abdullah, B. A., Yousef, A. H., Member, S. and Wahba, A. M. (2016) "Enhanced Code Conversion Approach for the Integrated Cross-Platform Mobile Development (ICPMD)", 42(11), p. 1036–1053.
- [6] Build native mobile apps using JavaScript and React. Available at: <https://facebook.github.io/react-native/> (Accessed: March 2019).
- [7] Platform Xamarin. Available at: <https://www.xamarin.com/platform> (Accessed: March 2019).
- [8] Apache Cordova. Available at: <https://cordova.apache.org> (Accessed: March 2019).
- [9] Malavolta, I., Ruberto, S., Soru, T. and Terragni, V. (2015) "Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation", p. 56–59. doi: 10.1109/MobileSoft.2015.15.
- [10] Bosnic, S., Papp, I. and Novak, S. (2016) "The development of hybrid mobile applications with Apache Cordova". doi: 10.1109/TELFOR.2016.7818919.
- [11] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé and J. Klein, "Automated Testing of Android Apps: A Systematic Literature Review," 2018 in IEEE Transactions on Reliability. doi: 10.1109/TR.2018.2865733
- [12] Zein, S., Salleh, N. and Grundy, J. (2016) "A systematic mapping study of mobile application testing techniques", Journal of Systems and Software. Elsevier Inc., 117, p. 334–356. doi: 10.1016/j.jss.2016.03.065.
- [13] Myers, G. J., Thomas, T. M. and Wiley, J. (2004) The Art of Software Testing.
- [14] Danglot, B., Vera Perez, O., Yu, Z., Monperrus, M., and Baudry, B. (2017) "A Snowballing Literature Study on Test Amplification". CoRR. Available at: <https://arxiv.org/abs/1705.10692>
- [15] Morgado, I. C. and Paiva, A. (2015b) "The iMPAcT Tool: Testing UI Patterns on Mobile Applications", 30th IEEE/ACM International Conference on Automated Software Engineering The, p. 876–881. doi: 10.1109/ASE.2015.96.
- [16] Zaeem, R. N., Prasad, M. R. and Khurshid, S. (2014) "Automated generation of oracles for testing user-interaction features of mobile apps", In: IEEE 7th International Conference on Software Testing, Verification and Validation, ICST 2014, p. 183–192. doi: 10.1109/ICST.2014.31.
- [17] Android Core App Quality. Available at: <https://developer.android.com/docs/quality-guidelines/core-app-quality> (Accessed: March 2019).
- [18] Adamsen, C. Q., Mezzetti, G. and Møller, A. (2015) "Systematic Execution of Android Test Suites in Adverse Conditions", In: The International Symposium on Software Testing and Analysis (ISSTA).
- [19] Morgado, I. C. and Paiva, A. (2015a) "Testing approach for mobile applications through reverse engineering of UI patterns", 2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), p. 42–49. doi: 10.1109/ASEW.2015.11.
- [20] Apple iOS Human Interface Guidelines. Available at: <https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/> (Accessed: March 2019).
- [21] Linares-vásquez, M., Bavota, G., Tufano, M., Moran, K., Penta, M. Di, Vendome, C., Bernal-cárdenas, C. and William, C. (2017) "Enabling Mutation Testing for Android Apps". doi: 10.1145/3106237.3106275.
- [22] Holl, K. and Elberzhager, F. (2014) "A Mobile-specific Failure Classification and its Usage to Focus Quality Assurance". doi: 10.1109/SEAA.2014.19.
- [23] Appium. Available at: <http://appium.io/> (Accessed: March 2019).

Algebraic Convergence to Software-Knowledge: Deep Software Learning (TSE)

Iaakov Exman and Assaf B. Spanier
Software Engineering Department
The Jerusalem College of Engineering – JCE - Azrieli
Jerusalem, Israel
iaakov@jce.ac.il, shpanier@jce.ac.il

Abstract— It is an empirical observation that Software Engineering and Knowledge Engineering seem to converge to a single discipline which may be suitably called *Software-Knowledge*. However, mere empirical observations are not satisfactory. These should be justified by plausible arguments. There are three convergence aspects, semantic, algebraic and topological, and this paper focuses on the algebraic aspect. Linear algebra is the basis for Linear Software Models, a rigorous theory of software systems composition from sub-systems, recently developed. Linear algebra, with added non-linearity, is also the basis for Deep Learning, a successful Artificial Intelligence domain. This work suggests and analyzes *Deep Software Learning*, i.e. Deep Learning specific to Software development problems. We then conjecture on deep reasons for Software-Knowledge convergence.

Keywords: *Linear Software Models; Software Composition; Laplacian Matrix; Deep Software Learning; Software-Knowledge Convergence; RNN; LSTM; Sequential and Structured Data.*

I. INTRODUCTION

It has been observed empirically that the Software Engineering and Knowledge Engineering disciplines seem to converge along their relatively short histories. Convergence is interesting theoretically and in practice. This paper analyzes Deep Software Learning as a mutual Software and Knowledge interaction for Software Development problems.

This work ultimate goal is to point out concrete directions to the rationale of Software-Knowledge convergence, starting from plausible conjectures. It offers a discussion roadmap for the Theory of Software Engineering special session on Software-Knowledge convergence, within the SEKE'2019 conference.

A. Concise Historical Overview

Software as a discipline starts in 1956 with Backus' Fortran a *high-level* programming language. Software Engineering itself was coined only in the celebrated NATO 1968 conference. Software history is since then a continuous increase in abstraction level of languages and design techniques. From structured programming, to object-oriented languages, as Java, to modeling languages, as UML, and model-driven-engineering

up to ontologies as conceptual refinement of classes and inheritance by subclasses.

Knowledge, an Artificial Intelligence (AI) field, started in 1950, with Shannon's [15], and Turing's [16] pioneer papers. Next appear classical AI expert-systems, e.g. Dendral to resolve chemical structural formulas. These systems separated inference engines from knowledge bases. Ontologies resulted from this research thread.

An early algebraic learning theory is the 1969 Perceptrons by Minsky and Papert [13]. A neural networks sub-field developed, remaining in research laboratories, for the lack of computing power. They were renamed "Deep Learning" with the industrial applications surge, due to added computing power (e.g. GPU), big data sets, and algorithmic improvements.

B. Aspects of Software-Knowledge Convergence

This paper is motivated by the following conjecture:

Software-Knowledge Conjecture

Software Engineering and Knowledge Engineering are converging to a single discipline which we call *Software-Knowledge*.

Software-Knowledge convergence consists of three aspects:

- 1) **Semantic** – the importance of concepts and ontologies in both software and knowledge fields;
- 2) **Algebraic** – mostly linear algebra as the basis of software composition theory and Deep Learning in diverse knowledge domains; this paper's focus;
- 3) **Topological** –graphs (planar or upon manifolds), with meaningful entities in nodes linked by edges.

Semantics got prominence within Software Engineering with the claim by Frederick Brooks in his books [1] [2] that "Conceptual Integrity is the most important consideration for software system design". This has been followed by recent research, e.g. by Jackson [8] and Exman. Semantics within Knowledge Engineering is prevalent since classical AI research, somewhat eclipsed by Deep Learning. Ontologies (e.g. the Protégé tool) are an important facet of it.

The Algebraic Software Engineering aspect, recognizing the importance of Software mathematical theory, e.g. that Linear Software Models [3], [4] gradually gains traction. The Algebraic Knowledge aspect, recognizing Deep Learning's applied surge.

II. RELATED WORK

A. Deep Learning for/by Software Engineering

Relevant neural networks are Recurrent Neural Networks (RNN), proposed in 1986 by Rumelhart, Hinton and Williams [14], and Long Short-Term Memory (LSTM) a special kind of RNN, proposed by Hochreiter and Schmidhuber [7] in 1997.

Software Engineering applications of Deep Learning for higher abstraction levels include: program generation from user intention (Lili Mou et al. [12]); program comprehension, to generate comments to Java code. (Xing Hu et al. [20]); API functions extraction from annotated code snippets collected from GitHub (Xiaodong Gu et al. [19]); and software modeling for various tasks (Hoa Khanh Dam et al. [6]).

Practical tools deal with software Traceability (Jin Guo et al. [11]), and fixing of C program errors (Rahul Gupta et al. [5]). Wei Fu and Tim Menzies [17] combine classical AI with Deep Learning to shorten training tasks.

B. Algebraic Software Theory: Linear Software Models

Software composition algebraic theory formalizes Brooks' Conceptual Integrity idea. Software is a hierarchical system, where each level is represented by a Modularity Matrix [3], [4]. Matrix columns stand for structural units, object-oriented *classes*, and matrix rows for functional units, i.e. class *methods*.

Brooks' principles translated into linear algebra demand that all matrix column vectors be *linearly independent* and similarly all the row vectors be linearly independent, obtaining a square matrix. If vector subsets are disjoint to other subsets, the matrix displays a block-diagonal form, i.e. the modules are *orthogonal*.

Modularity matrices may have outliers coupling between modules. Spectral methods for the Modularity Matrix [3], or the respective Laplacian Matrix [4], resolve couplings. A Laplacian obtains the same modules as the Modularity Matrix. The Fiedler vector, fitting the lowest Laplacian non-zero eigenvalue, allows locating outliers and splitting of too sparse modules.

III. DEEP SOFTWARE LEARNING: THE PROBLEMS

Deep Software Learning has to assume that software is a collection of diverse assets: requirements, class diagrams, statecharts for design, a variety of graphs, models and code.

A. Software Problems to be Solved

Software problems dealt with by Deep Learning can be classified by their abstraction levels (Fig. 1). Higher abstraction activities, such as API Extraction, Program Generation and Program Comprehension depend on a suitable underlying *software modeling*. Modeling is high-level abstraction, since many activities involve *translation between models*.

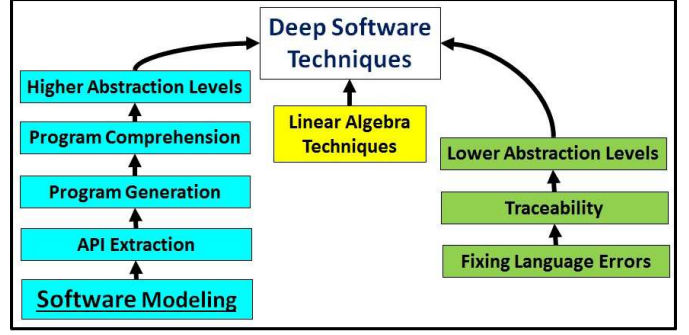


Figure 1. Deep Learning to Software Engineering applications, classified by abstraction levels. In between, the essential Linear Algebra Techniques.

Even lower abstraction level activities, such as correcting program errors, as done in DeepFix [5] need modeling. Every programmer has experience with accumulated bugs that result from misinterpretation (by the compiler!) of only a few bugs.

Linear algebra techniques are essential for Deep Learning. Richard Wei et al. [18], in their Compiler Infrastructure for Deep Learning, emphasize linear algebra representation in their system, such as a first class tensor type, algebraic operators such as “dot” and “tanh” (a typical sigmoid-like activation function).

B. Software Characterization: Sequential but not Consecutive, and Structured

Often program feature pairs are sequential but not appearing in each other neighborhood. Examples are: left and right parentheses (or braces); open and close a file; Java try and catch. Code modeling is sequential, but not of consecutive tokens. Dealing with such sequences, demands specific Deep Learning (DL) networks. Software also has more complex structures such as abstract syntax trees, dependency graphs, design diagrams. Sequences are not enough for software DL.

IV. RECURRENT NEURAL NETWORKS

Recurrent Neural Networks (RNNs) are dedicated to continuous data, such as text, audio and video. It reuses previous information about a word in a sentence or video frame to understand the next word or frame. RNNs handle tasks, like free text comprehension and text sequences generation from scratch. To reuse previous information to handle the next input, RNN has recourse to persistence loops. RNNs have difficulty applying prediction of long-distance natural language dependencies. Most translation, voice recognition, and image classification successes, are due to LSTM a special class of RNNs.

A. LSTM = Long Short-Term Memory

LSTMs remember information for long periods. Forgetting must be explicitly handled. LSTMs also have a loop structure of repeating neural network units. The main difference of a typical LSTM unit from an RNN unit, is 4 layers instead of a single one. A hidden state Z is the key to LSTMs functioning. It has an input z_{t-1} from its predecessor, runs throughout the chain of (unrolled loop) units, affected by controlled interactions, and outputs z_t to its successor unit. Three gates (in Fig. 2) update the hidden state Z in each cycle, filtering the output Y parts:

- **Forget gate f_t** – sigmoid taking y_{t-1} and x_t and producing a number between 0 and 1 for each z_{t-1} value; the part of the hidden state Z to (fully or partially) discard;
- **Input gate** – has 2 layers: *sigmoid* i_t sets which values will be updated; *tanh* creates a new vector of values \hat{z}_t , multiplied by the sigmoid output giving candidate values actually added to Z: $\hat{z}_t = f_t * z_{t-1} + i_t * \hat{z}_t$
- **Output gate** – sigmoid o_t filters what will be the output and what remains the Z output: *tanh* normalizes z_t values between -1 and +1, then multiplied by the sigmoid o_t :
 $y_t = o_t * \tanh(z_t)$.

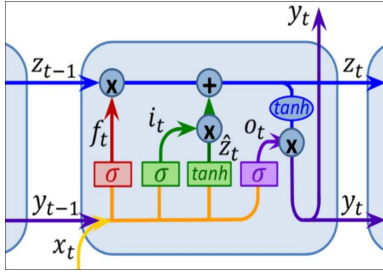


Figure 2. LSTM Deep Learning Network Schematic diagram - The upper (blue) horizontal line is the Z hidden state, with input z_{t-1} and hidden output z_t . The lower horizontal line passes the output y_t between consecutive units. The three vertical gates are: the (red) forget gate f_t ; the (green) input gate i_t with two parallel layers (σ and \tanh); the (violet) output gate o_t . (Color online)

A realistic LSTM test, keeping track of long-range attributes, takes a large source code and randomly concatenates it into a long file (e.g. Linux Kernel [9] about $6 \cdot 10^6$ characters). The LSTM network is trained to predict special source code symbols, as whitespace, quotation marks and brackets. To predict a *close* bracket, the model must be aware of a matching *open* bracket, appearing many time steps ago. LSTM performs much better than other learning models due to an additional state feature (hidden state Z) explained above, in contrast to the standard RNN single hidden state. Results can be improved with an attention mechanism [21] or bidirectional-LSTM [10].

V. DEEP SOFTWARE LEARNING IN PRACTICE

The Deep Learning (DL) mechanism in the Xing Hu [20] work deals with sequential code, and Abstract Syntax Tree (AST) structure, translating sequence-to-sequence, code to comments. The Encoder/Decoder (both LSTMs) architecture uses one Encoder pass to traverse the AST, learning the code relevant to the comments. A second Encoder pass composes the gathered comment pieces into meaningful sentences.

Rare tokens clutter a vocabulary with single instances. Practical projects exchange numerals/strings by generic tokens <NUM> and <STR>, and rare words by “unknowns” <UNK>.

VI. DISCUSSION

Today’s software and knowledge (DL) theories have two important characteristics: 1- they heavily involve linear algebra; 2- the algebra is totally independent of concepts’ semantics.

Remaining issues are: The algebraic software theory is up to now strictly linear, while Deep Learning involves non-linearity. Will there be a convergence also in this sense? The Laplacian matrix is central to the software theory, while not so prominent in Deep Learning; will it be important for Deep Learning too?

References

- [1] F.P. Brooks, *The Mythical Man-Month, Essays on Software Engineering*, Anniversary Edition, Addison-Wesley, Boston, MA, USA, (1995).
- [2] F. Brooks, *The Design of Design, Essays from a Computer Scientist*, Addison-Wesley, Boston, MA, USA, 2010.
- [3] I. Exman, “Linear Software Models: Decoupled Modules from Modularity Eigenvectors”, *Int. Journal on Software Engineering and Knowledge Engineering*, vol. 25, pp. 1395-1426, October 2015. DOI: [10.1142/S0218194015500308](https://doi.org/10.1142/S0218194015500308)
- [4] I. Exman and R. Sakhnini, “Linear Software Models: Bipartite Isomorphism between Laplacian Eigenvectors and Modularity Matrix Eigenvectors”, *Int. Journal of Software Engineering and Knowledge Engineering*, Vol. 28, No 7, pp. 897-935, 2018. DOI: [http://dx.doi.org/10.1142/S0218194018400107](https://doi.org/10.1142/S0218194018400107)
- [5] R. Gupta, S. Pal, A. Kanade and S. Shevade, “DeepFix: Fixing Common C Language Errors by Deep Learning”, *Proc. 31st AAAI Conference*, pp. 1345-1351, (2017).
- [6] Hoa Khanh Dam, Truyen Tran and Trang Pham, “A deep language model for software code”, *arXiv:1608.02715*, (August 2016).
- [7] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural Computation*, Vol. 9, pp. 1735-1780, (1997).
- [8] D. Jackson, “Conceptual Design of Software: A Research Agenda”, *MIT-CSAIL-TR-2013-020*, August 8, 2013.
- [9] A.Karpathy, J. Johnson and L. Fei-Fei, “Visualizing and Understanding Recurrent Networks”, <https://arxiv.org/pdf/1506.02078.pdf>.
- [10] E. Kiperwasser and Y. Goldberg, “Simple and accurate dependency parsing using bidirectional LSTM feature representations”, *Trans. Assoc. Computational Linguistics*, Vol. 4, pp. 313-327, (2016).
- [11] Jin Guo, Jinghui Chang and Jane Cleland-Huang, “Semantically Enhanced Software Traceability Using Deep Learning Techniques”, *arXiv:1804.02438* (April 2018).
- [12] Lili Mou, Rui Men, Ge Li, Lu Zhang and Zhi Jin, “On End-to-End Program Generation from User Intention by Deep Neural Networks”, *arXiv:1510.07211*, (October 2015).
- [13] M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA, USA, 1969.
- [14] D.E. Rumelhart, G.E. Hinton and R.J. Williams, “Learning internal representations by error propagation”, in Rumelhart and McClelland (eds.) *Parallel Distributed Processing*, vol. 1, pp. 318-362, MIT Press, Cambridge, MA, USA, (1986).
- [15] C. E. Shannon, “Programming a Computer for Playing Chess”, *Phil. Mag.*, Series 7, Vol. 41, 18 pages (March 1950).
- [16] A.M. Turing, “Computing Machinery and Intelligence”, *Mind*, New Series, Vol. 59, pp. 433-460, (October 1950).
- [17] Wei Fu and Tim Menzies, “Easy over Hard: A Case Study on Deep Learning”, *ESEC/FSE’17*, *arXiv:1703.00133*, (June 2017). DOI: <https://doi.org/10.1145/3106237.31052546>
- [18] R. Wei, L. Schwartz and V. Adve, “DLVM: A Modern Compiler Infrastructure for Deep Learning Systems, Workshop track ICLR 2018”, *arXiv:1711.03016*, (February 2018).
- [19] Xiaodong Gu, Hongyu Zhang, Dogmei Zhang and Sunghun Kim, “Deep API Learning”, *Proc. FSE’16*, *arXiv:1605.08545* (2017). DOI: [http://dx.doi.org/10.1145/1235](https://doi.org/10.1145/1235)
- [20] Xing Hu, Ge Li, Xin Xia, David Lo and Zhi Jin, “Deep Code Comment Generation”, *Proc. ICPC IEEE/ACM Int. Conf. Program Comprehension*, 11 pages (May 2018). DOI: https://doi.org/10.475/123_4
- [21] W. Yin, H. Schutze, B. Xiang and B. Zhou “Abcnn: Attention-based convolutional network for modeling sentence pairs”, *Trans. Assoc. Computational Linguistics*, Vol. 4, pp. 259-272 (2016).

Formal ontologies and data shapes within the Software Engineering development lifecycle (TSE)

Jose María Álvarez Rodríguez

Department of Computer Science and Engineering
Carlos III University of Madrid
Madrid, Spain
josemaria.alvarez@uc3m.es

Valentín Moreno, Juan Llorens

Department of Computer Science and Engineering
Carlos III University of Madrid
Madrid, Spain
{vmpelayo,llorens}@inf.uc3m.es

Abstract— Business models, organizational activities and corporate strategies are now being reshaped to meet the new needs of a challenging and evolving environment in which more up-to-date, secure, safer, cost-efficient and personalized software products and services must be timely delivered. This new digital context also represents an opportunity for the improvement and extension of existing software engineering methods. One of the current trends in Software Engineering development lies in boosting interoperability and collaboration between tools and people through the sharing of existing artifacts under common data models, formats and protocols to improve the practice and reuse of existing software artifacts. In this context, formal ontologies and data shapes play a key role to model and exchange data and to provide services for data validation (consistency checking) or type inference as part of a knowledge management strategy. In this document, an initial review of the different approaches to model and exchange data of software artifacts is done to finally evaluate and discuss the proper mechanisms to technically support the upcoming needs in the Software Engineering development lifecycle.

Keywords: *software development lifecycle; ontologies; data shapes; interoperability; knowledge representation*

I. INTRODUCTION

The Digital Age, the “Society 5.0” in Japan or the “4th Industrial Revolution” [1] has come to say that any industry, business or even our daily life will suffer a deep transformation being driven by software systems.

This new digital context also represents a challenge and an opportunity for the improvement and extension of existing software engineering methods. After 50 years [2], the Software Engineering (SE) discipline has focused on ensuring the efficiency, correctness, robustness and reliability of software systems. The development of some SE knowledge areas [3] have set the foundations of the discipline. Furthermore, the definition of methodologies, methods and models, software development lifecycles and their technological support have also represented a major step to improve the SE practice from both organizational and scientific/technological points of view[4] [5].

On the other hand, knowledge management techniques have gained enough momentum in the SE discipline to elevate the meaning of the implicit knowledge coded into software pieces. Software as a knowledge asset is becoming a commodity that is embedded in products, business or manufacturing processes. It

is a new kind of intellectual asset that can help us to reduce costs and time to market, and to generate competitive advantage. In this light, knowledge management techniques [6] can be applied to capture, structure, store and disseminate software artifacts to directly support methods such as software reuse.

However, one of the cornerstones in knowledge management lies in the selection of an adequate representation paradigm. After a long time [7], this problem still persists since a suitable representation format (and syntax) can be reached in several ways.

Obviously, different types of knowledge, such as those in the SE discipline (requirements, source code, test cases, etc.), require different types of representation [8] [9]. But, on the other hand, knowledge management also implies the standardization of data and information within the development lifecycle. Any bit of information must be structured and stored for supporting other application services such as business analytics or knowledge discovery.

One of the current trends in SE development lies in boosting interoperability and collaboration between tools and people through the sharing of existing artifacts under common data models, formats and protocols to improve the practice and reuse of existing software artifacts. In this context, OSLC (“Open Services for Lifecycle Collaboration”), model-driven approaches, etc. are defining a collaborative software development ecosystem [10] through the definition of data shapes that serve us as a contract to get access to information resources through standardized.

In particular, the Representational State Transfer (REST) software architecture style is commonly used to manage information and software resources such as requirements, test cases or even source code that are publicly represented and exchanged in standard formats such as RDF or just XML. Obviously, these approaches represent a big step towards the integration and interoperability between the agents involved in the software development lifecycle.

However, a knowledge management strategy to take advantage of software artifacts is beyond of the mere exchange of data. Consistency checking, type inference, data integrity, etc. are common operations expected in a knowledge-centric framework that can help us to improve the practice in the SE discipline.

In this paper, authors review and discuss the role of ontologies and data shapes as a mean to represent, exchange and consume software-related artifacts with the aim of improving and enriching software development methods providing capabilities for consistency checking, type inference or data integrity.

II. BACKGROUND

The Software Engineering discipline [11] [12] [13] initially focused on the definition of methodologies and methods to tackle the problem of reliability in software systems and to ease the reuse of working software products.

The notion of software engineering process was then defined, and different software development lifecycles (SDLC) such as the Structured Systems Analysis and Design Method (SSADM), the Waterfall model, the Rational Unified Process (RUP), the Rapid application development (RAD) or the Vee model were established as a method to manage the complexity of software development. Afterwards, new programming paradigms, standard notations (e.g. UML or OCL), languages and tools emerged as way of improving the abstract thinking in combination with a technological support providing better development environments.

Once, the methods and the supporting tools were available, the focus was the reuse of software assets, the quality management, the definition of maturity models and the automation of tasks generated during the software development process. More specifically, the software reuse area gained momentum through the definition of methods to reuse existing software components through the abstract description of different elements, e.g. architectural and design patterns, libraries, component models or services. Furthermore, coding practices [14] were also improved adding new foundations such as the SOLID¹ principles or new practices such as refactoring.

Initiatives such as Model-Driven Engineering [15] [16] (MDE) and Model-Driven Architecture (MDA) later posed the foundations to automate the production of software for specific domains. In the last decade, software product lines [17] have also been subject of study and application as a method for automating the creation of families of software products.

At the development level, the emerging use of Agile methods [18], inspired by the Spiral model, such as XP, Scrum or Kanban and, processes such as BDD (Behavior-Driven Development) or TDD (Test Driven-Development) have also led us to improve the software development practice in combination with approaches such as DevOps [19] (Development and Operations). The latter are often applying to ease the continuous transition of software systems from a development to a production environment as a mean for the early detection of “bugs” and the improvement of the maintenance and change management processes.

In summary, the Software Engineering discipline has reached a great maturity level. There is a huge body of

knowledge and practice that allow us to acknowledge which are, and which are not, the software engineering practices that really represent timeless scientific and technological foundations for a proper software development process. That is why, knowledge management techniques are aimed at enhancing existing SE methods by providing a knowledge layer on top of the data generated during the development lifecycle.

III. FORMAL ONTOLOGIES VS DATA SHAPES

In the early days of the Semantic Web, formal ontologies [20] designed in RDFS (Resource Description Framework Schema) or OWL were the key technologies to model and share knowledge.

From upper ontologies such as DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) or SUMO (The Suggested Upper Merged Ontology) to specific vocabularies such FOAF (Friend Of A Friend) or SKOS (Simple Knowledge Organization System), the process to share knowledge consisted in designing a formal ontology for a particular domain and populate data (instances) for that domain. Although the complete reuse of existing ontologies was expected, the reality demonstrated that every party willing to share knowledge and data would create its own ontologies. Thus, the main idea behind web ontologies was partially broken since just a few concepts were really reused.

Once the Linked Data initiative (RDF + HTTP) emerged to unleash the power of existing databases, a huge part of the Semantic Web community realized that a formal ontology was not completely necessary to exchange data and knowledge.

Taking into account that ontologies were still present, the efforts were focused on providing methods for data consistency [21] through the execution of procedures such as: 1) reasoning processes to check consistency, and 2) rules, mainly in SWRL (Semantic Web Rule Language) or SPARQL [22] [23]. These procedures are not fully recommended, due to performance issues, when a huge number of instances are available. As a new evolution, then, the community realized that ontology-based reasoning was not the most appropriate method for data validation when data is being exchanged.

That is why in recent times the Semantic Web community has seen an emerging interest to manage and validate RDF datasets according to different shapes and schemes. A data shape can be defined as a resource, i.e. metamodel, that describes contents of, and constraints on, other resources. In this way, it is possible to not just improve but to overcome some of the well-known restrictions [24] of RDF encoded data: 1) lack of support to represent certain knowledge features N-ary relationships [25], 2) practical issues dealing with reification [26] and blank nodes [27]. New specifications and methods for data validation (consistency) are being designed to turn reasoning-based validation into a kind of grammar-based validation.

¹ “The Principles of OOD”. Source: <http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod> (last visited: 20th of February, 2019)

Process	Type	Creation	Scope	Refs.
Consistency check	Vocabulary-based	Semantic Web reasoner	RDF datasets	[21]
Data validation (integrity)	Query-based	Hand-made RDF templates	RDF datasets	[22]
	Vocabulary-based	Hand-made	RDF datasets	[23]
	Vocabulary-based	Hand-made or automatically generated by an OSLC API	OSLC Resource Shape	[28]
	Vocabulary-based	Hand-made	Dublin Core Description Set Profiles	[29]
	Query-based	RDF Unit (test creation)	RDF datasets	[30]
	Query-based (generated from ShEX expressions)	Automatic generation of SPARQL queries	RDF datasets	[31] [32] [33]
	Vocabulary and Query-based	Automatic generation of SPARQL queries	OWL and RDF under Closed World Assumption	[34]
	Query-based	SPIN language + SPARQL queries	RDF datasets	[35]

Table 1 Comparison of methods for RDF data validation.

These methods take inspiration from existing approaches in other contexts such as DTD (Document Type Definition), XML-Schema or Relax NG (REgular LAnguage for XML Next Generation) for XML, or DDL (Data Definition Language) for SQL (Structured Query Language).

The W3C launched in 2014 the RDF Data Shapes Working group having as main outcome the SHACL (Shapes Constraint Language), W3C Recommendation, and the ShEX (Shape Expressions) language [31] [32]. Both are formal languages for expressing constraints on RDF graphs including cardinality constraints as well as logical connectives for disjunction and polymorphism. OSLC Resource Shapes [28], Dublin Core Description Set Profiles [29], and RDF Unit [30] were also other constraint languages for data validation in the context of Linked Data.

Following a more classical approach for RDF data validation, the Pellet Integrity Constraints is an extension of the existing semantic web reasoner [34] that interprets OWL ontologies under the Closed World Assumption with the aim of detecting constraint violations in RDF data. These restrictions are also automatically translated into SPARQL queries. This approach has been implemented on top of the Stardog² database, enabling users to write constraints in SPARQL, SWRL or as OWL axioms. This approach has been reported as a method for data validation and type inference in some case studies (e.g. NASA Knowledge Graph)³ were logical models are translated into OWL instances and, then, a reasoning process (semantic web based or rule-based) is executed to find inconsistencies, infer types, etc. However, the complexity and time to make transformations between object models and Description Logics seems to be not very effective since the same information is being modelled at the same time under two different paradigms. Finally, the SPIN language [35] also makes use of SPARQL (mainly its syntax) to define constraints on RDF-based data that can be executed by systems supporting SPIN (SPARQL Inferencing Notation), such as the TopBraid's toolchain.

In conclusion, the relevance of data validation to exchange RDF-encoded data is clear. RDF Data Shapes in its different flavors, such as OSLC Resource Shapes, are becoming the

cornerstone for boosting interoperability among agents, see Table 1. It is also clear that ontologies are becoming less important although a combined approach (data shapes and a formal ontology) can provide important benefits in terms of data validation and knowledge inference (if needed). In the context of SE and reuse, as it has been previously outlined, software artifacts must take advantage of new technologies to enable practitioners the automatic processing of exchanged data.

IV. KNOWLEDGE MANAGEMENT WITHIN THE SOFTWARE ENGINEERING PROCESS

Software is becoming a commodity, knowledge that is embedded in every product, business and development or manufacturing process. Assuming that a software artifact is a knowledge asset, the ground truth about knowledge characteristics [36] is also valid for software artifacts:

- Use of knowledge does not consume it.
- Transfer of knowledge does not imply losing it.
- Knowledge is abundant; the problem lies on the proper use and exploitation.
- Much of an organization's valuable knowledge walks out the door at the end of the day.

According to the current SE context, it seems that graph-approaches based on a semantic network and deployed under a set of standards in a service-oriented environment, are the most appropriate candidates. Considering this environment, the following knowledge representation paradigms (focusing on web-oriented technologies) have been selected for comparison:

1-The **Resource Description Framework** (RDF) [37] is a framework for representing information resources in the Web using a directed graph data model. The core structure of the RDF abstract syntax is a set of triples, each consisting of a subject, a predicate and an object. A set of such triples is called an RDF graph. An RDF graph can be visualized as a set of nodes and directed-arcs diagram, in which each triple is represented as a node-arc-node link. RDF has been used as the underlying data model for building RDFS/OWL ontologies, gaining momentum in the web-based environment due to the explosion of the Semantic Web and Linked Data initiatives that

² <http://stardog.com/>

³ <https://www.stardog.com/blog/nasas-knowledge-graph/>

aim to represent and exchange data (and knowledge) between agents and services under the web-based protocols.

2-The **RDF Schema (RDFS)** [38, p. 1] provides a data-modeling vocabulary for RDF data. It represents a first try to support the creation of formal and simple ontologies with RDF syntax. RDFS is a formal and simple ontology language in which it is possible to define class and property hierarchies, as well as domain and range constraints for properties. One of the benefits of this property-centric approach is that it allows anyone to extend the description of existing resources.

3-The **OWL (Ontology Web Language)** [39] is an ontology language for capturing meaningful generalizations about data in the Web. It includes additional constructors for building richer class and property descriptions (vocabulary) and new axioms (constraints), along with a formal semantics. OWL 1.1 consists of three sub-languages with different levels of expressivity: 1) OWL Lite, 2) OWL DL (Description Logics) and 3) OWL Full.

4-The **OWL 2.0** [39] family defines three different profiles: OWL 2 EL (Expressions Language), OWL 2 QL (Query Language) and OWL RL (Rule Language). These profiles represents a syntactic restriction of the *OWL 2 Structural Specification* and more restrictive than OWL DL. The use of profiles is motivated by the needs of different computational processes. OWL EL is designed for enabling reasoning tasks in polynomial time. The main aim of OWL 2 QL is to enable conjunctive queries to be answered in LogSpace using standard relational database technology. Finally, OWL 2 RL is intended to provide a polynomial time reasoning algorithm using rule-extended database technologies operating directly on RDF triples. In conclusion, OWL 2.0 adds new functionalities regarding OWL 1.x. Most of them are syntactic sugar but others offer new expressivity [39]: keys, property chains, richer datatypes, data ranges, qualified cardinality restrictions, asymmetric, reflexive, and disjoint properties; and enhanced annotation capabilities.

3-The **RIF Core** (Rule Interchange Format) [40] comprises a set of dialects to create a standard for exchanging rules among rule systems, in particular among Web rule engines. RIF was designed for exchanging rather than developing a single one-fits-all rule language. RIF dialects fall into three broad categories: first-order logic, logic-programming, and action rules. The family of dialects comprises: 1) logic-based dialects (RIF-BLD) including languages that employ some kind of logic such as First Order Logic (usually restricted to Horn Logic) or non-first-order logics; 2) rules-with-actions (RIF-PRD) dialects comprising rule systems such as Jess, Drools and JRules as well as event-condition-action rules such as Reaction RuleML. RIF also defines compatibility with OWL and serialization using RDF.

5-The **SRL** (System Representation Language) [41] [42] is based on the ground idea that whatever information can be described as a group of relationships between concepts. Therefore, the leading element of an information unit is the relationship. For example, Entity/Relationship data models are certainly represented as relationships between entity types;

software object models can also be represented as relationships among objects or classes; in the process modeling area, processes can be represented as causal/sequential relationships between sub-processes. Moreover, UML (Unified Modeling Language) or SysML (System Modeling Language) metamodels can also be modeled as a set of relationships between metamodel elements.

SRL also includes a repository model to store information and relationships with the aim of reusing all kind of knowledge chunks. Furthermore, text-based information can certainly be represented as relationships between terms by means of the same structure. Indeed, to represent human language text, a set of well-constructed sentences, including the subject+verb+predicate (SVP) should be used. The SVP structure can be then considered as a relationship typed V between the S and the predicated P. In SRL, the simple representation model for describing the content of whatever artifact type (requirements, models, tests, maps, text docs or source code) is as follows:

SRL representation for artifact $\alpha = i_\alpha = \{(RSHP_1), (RSHP_2), \dots, (RSHP_n)\}$ where every single RSHP (relationship) is called RSHP-description and must be described using terms.

One important consequence of this representation model is that there is no restriction to represent a particular type of knowledge. Furthermore, SRL has been used as the underlying information model to build general-purpose indexing and retrieval systems, domain representation models [41], approaches for quality assessment of requirements and knowledge management tools such as knowledgeMANAGER.

Obviously, a plethora of other knowledge representation mechanisms and paradigms can be found as it is presented below. However, we focus here on comparing those that satisfy the three basic ideas of this study: 1) a language for representing any artifact metadata and contents; 2) a system for indexing and retrieval and 3) a standard input/output interface (data shape+REST+RDF) to share and exchange artifact metadata and contents.

The SBVR (Semantics of Business Vocabulary and Rules). It is an OMG standard to define the basis for formal and detailed natural language declarative description of a complex entity. The Ontology Definition Metamodel (ODM). It is an OMG standard for knowledge representation, conceptual modeling, formal taxonomy development and ontology definition. It enables the use of a variety of enterprise models as starting points for ontology development through mappings to UML and MOF.

ODM-based ontologies can be used to support: 1) interchange of knowledge; 2) representation of knowledge in ontologies and knowledge bases; and 3) specification of expressions that are the input to, or output from, inference engines. The Reusable Asset Specification (RAS), an OMG standard that addresses the engineering elements of reuse. It

attempts to reduce the friction associated with reuse transactions through consistent, standard packaging.

V. EVALUATION AND DISCUSSION

The previous section has reviewed the main approaches for knowledge representation in a web-oriented environment for exchanging software artifacts. As a result, Table 2 (in the Annex) shows the main characteristics and capabilities that can be found in RDF, RDFS, OWL and SRL with special focus on those regarding knowledge management and, more specifically knowledge representation. In order to select the proper mechanism for knowledge representation of software artifacts, the following points must be considered:

- RDF is based on a directed graph and it can only represent binary relationships (unless reification and blank nodes are used). As a representation mechanism, RDF presents some restrictions that have been outlined in several works [24]. For instance, N-ary relationships [25], practical issues dealing with reification [26] and blank nodes [27] are well-known RDF characteristics that do not match the needs of a complete framework for knowledge representation. Furthermore, RDF is built on two main concepts: resources and literals. However, a literal value cannot be used as the subject of an RDF triple. Although this issue can be overcome using a blank node (or even reification) and the property `rdf:value`, it adds extra complexity for RDF users. Finally, RDF has been designed to represent logical statements, constraining also the possibility of representing other widely used paradigms such as objects or entity-relationships models. Due to these facts, it seems clear that RDF can be used for exchanging data, but it is not the best candidate for knowledge representation.
- RDFS is a good candidate for modeling lightweight formal ontologies including some interesting capabilities close to object-oriented models. RDFS ontologies can be serialized as RDF, but this feature can also be a disadvantage due to expressivity restrictions of RDF. Again, RDFS has been designed for expressing logical statements that describe web resources, so its use for other types of information seems to be not advisable.
- Building on the previous discussion, OWL presents a family of logic dialects for knowledge representation. It is based on strong logic formalisms such as Description Logic or F-Logic. It was also designed for asserting facts about web resources although it can be used as a general logic framework for any type of knowledge. One of the main advantages of OWL is the possibility of performing reasoning processes to check consistency or infer types. However, reasoning can be considered harmful in terms of performance and most of times it is not necessary when data is being exchanged. Besides, OWL is not the best candidate for data validation, a key process in knowledge exchange.
- RIF Core and the family of RIF dialects have been included in this comparison due to the fact that most of

domain knowledge is embedded in rules. However, RIF was not designed for data validation and its acceptance is still low (just a few tools export RIF and less are capable of importing RIF files). On the other hand, RIF makes use of the web infrastructure to exchange rules, what means also that this environment is a very good candidate to exchange data, information and knowledge.

- SRL, based on relationships, allows domain experts to create relationships between terms, concepts or even artifacts (containers). It provides a framework for knowledge representation with capabilities for expressing any kind of cardinality and N-ary relationships. SRL is based on undirected property graphs, enhancing expressivity. Although it has not been directly designed for data validation, its metamodel allows the possibility of checking cardinality, value constraints, domain and range restrictions. One of the strong points of SRL is the native support of a tool such as knowledgeMANAGER and the possibility of automatically providing semantic indexing and retrieval mechanisms. Both have generated a relevant acceptance in the industry for authoring and quality checking.

Finally, as a general comment, there is also a lack of tools working natively on RDF. Furthermore, RDF was conceived to exchange information over the web. Although some RDF repositories can provide capabilities for indexing and searching RDF resources through an SPARQL interface, the experience has demonstrated that most of times RDF is translated into the native data model of a tool.

Based on this evaluation and considering the three basic requirements of this review, we conclude that RDF is a good alternative to exchange data. Since formal ontologies and reasoning processes are not completely necessary and, instead, data validation is a key aspect for boosting interoperability and reuse of software artifacts, it also seems clear that an approach like SRL can perfectly fit to the major objective of knowledge representation in the SE discipline. However, we recognize that the use of formal RDFS or OWL ontologies is not incompatible with data shapes, but possible. RDFS and OWL are languages for building domain vocabularies, while SRL is already a domain vocabulary for knowledge representation, so it is possible to define SRL in a formal RDFS/OWL ontology.

VI. CONCLUSIONS AND FUTURE WORK

The application of the Linked Data principles to exchange data in the software development lifecycle is gaining momentum. Software artifacts reuse via interoperability is a key enabler for boosting collaboration in the development of complex software systems. The concept of continuous engineering is becoming a reality since it is possible to integrate data and services under common protocols and data models.

In this context, the capability of reusing existing software artifacts is a key factor that can ease teams to develop systems faster and safer. However, software is not anymore, a piece of logical instructions but a kind of knowledge and organizational asset. That is why a proper environment for knowledge

management of software artifacts should provide the appropriate mechanisms for representing, storing, indexing and retrieving any kind of software artifact. However, some approaches such as OSLC relying on Linked Data principles cannot easily be deployed due to issues regarding expressivity in RDF or the need of tools for easily processing RDF (mainly from a developer perspective). In this context, the use of well-documented REST APIs (e.g. based on standards such as the OpenAPI Specification) are good enough to access and exchange data generated during the development lifecycle.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the Celtic Next-EUREKA initiative under code N° C2017/3-2-IoD (Internet of DevOps) and from specific national programs and/or funding authorities.

REFERENCES

- [1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Bus. Inf. Syst. Eng.*, vol. 6, no. 4, pp. 239–242, 2014.
- [2] M. Kersten, "Five Predictions for the Coming Decades of Software," *IEEE Softw.*, vol. 35, no. 5, pp. 7–9, Sep. 2018.
- [3] P. Bourque, R. E. Fairley, and others, *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.
- [4] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *Ieee Softw.*, vol. 29, no. 6, pp. 18–21, 2012.
- [5] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *J. Syst. Softw.*, vol. 86, no. 6, pp. 1498–1516, 2013.
- [6] I. Nonaka and H. Takeuchi, *The knowledge-creating company: How japanese companies create the dynamics of innovation*. New York: Oxford University Press, 1995.
- [7] R. Hull and R. King, "Semantic database modeling: Survey, applications, and research issues," *ACM Comput. Surv. CSUR*, vol. 19, no. 3, pp. 201–260, 1987.
- [8] R. Davis, H. Shrobe, and P. Szolovits, "What is a knowledge representation?," *AI Mag.*, vol. 14, no. 1, p. 17, 1993.
- [9] T. Groza, S. Handschuh, T. Clark, S. Buckingham Shum, and A. de Waard, "A short survey of discourse representation models," 2009.
- [10] K. Manikas and K. M. Hansen, "Software ecosystems – A systematic literature review," *J. Syst. Softw.*, vol. 86, no. 5, pp. 1294–1306, May 2013.
- [11] R. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. New York, NY, USA: McGraw-Hill, Inc., 2010.
- [12] I. Sommerville, *Software engineering*, Tenth edition. Boston: Pearson, 2016.
- [13] C. Ebert, "50 Years of Software Engineering: Progress and Perils," *IEEE Softw.*, vol. 35, no. 5, pp. 94–101, Sep. 2018.
- [14] R. C. Martin, Ed., *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ: Prentice Hall, 2009.
- [15] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, Feb. 2006.
- [16] M. Brambilla, J. Cabot, and M. Wimmer, "Model-Driven Software Engineering in Practice," *Synth. Lect. Softw. Eng.*, vol. 1, no. 1, pp. 1–182, Sep. 2012.
- [17] P. Clements and L. Northrop, *Software product lines: practices and patterns*. Boston: Addison-Wesley, 2002.
- [18] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.
- [19] L. Bass, I. M. Weber, and L. Zhu, *DevOps: a software architect's perspective*. New York: Addison-Wesley Professional, 2015.
- [20] V. R. Benjamins, D. Fensel, and A. Gómez-Pérez, "Knowledge Management through Ontologies," in *PAKM*, 1998.
- [21] K. Baclawski, M. M. Kokar, R. J. Waldinger, and P. A. Kogut, "Consistency Checking of Semantic Web Ontologies," in *International Semantic Web Conference*, 2002, pp. 454–459.
- [22] C. Bizer and R. Cyganiak, "Quality-driven information filtering using the WIQA policy framework," *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 7, no. 1, pp. 1–10, Jan. 2009.
- [23] A. Hogan, J. Umbrich, A. Harth, R. Cyganiak, A. Polleres, and S. Decker, "An empirical survey of Linked Data conformance," *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 14, pp. 14–44, Jul. 2012.
- [24] S. Powers, *Practical RDF*. Beijing; Sebastopol: O'Reilly, 2003.
- [25] N. Noy and A. Rector, "Defining N-ary Relations on the Semantic Web," W3C Working Group, 2006.
- [26] V. Nguyen, O. Bodenreider, and A. Sheth, "Don't like RDF reification?: making statements about statements using singleton property," 2014, pp. 759–770.
- [27] A. Mallea, M. Arenas, A. Hogan, and A. Polleres, "On blank nodes," in *The Semantic Web—ISWC 2011*, Springer, 2011, pp. 421–437.
- [28] A. G. Rymann, A. L. Hors, and S. Speicher, "OSLC Resource Shape: A language for defining constraints on Linked Data," in *LDOW*, 2013.
- [29] K. Coyle and T. Baker, "Dublin Core Application Profiles Separating Validation from Semantics," W3C, Mar. 2013.
- [30] D. Kontokostas *et al.*, "Test-driven evaluation of linked data quality," in *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, 2014, pp. 747–758.
- [31] I. Boneva, J. E. L. Gayo, S. Hym, E. G. Prud'hommeau, H. R. Solbrig, and S. Staworko, "Validating RDF with Shape Expressions," *CoRR*, vol. abs/1404.1270, 2014.
- [32] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva, and D. Kontokostas, "Validating RDF Data," *Synth. Lect. Semantic Web Theory Technol.*, vol. 7, no. 1, pp. 1–328, Sep. 2017.
- [33] J. Alvarez-Rodríguez, J. Labra-Gayo, and P. Ordoñez de Pablos, "Leveraging Semantics to Represent and Compute Quantitative Indexes: The RDFIndex Approach," in *Metadata and Semantics Research*, vol. 390, E. Garoufallou and J. Greenberg, Eds. Springer International Publishing, 2013, pp. 175–187.
- [34] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *J Web Sem*, vol. 5, no. 2, pp. 51–53, 2007.
- [35] Holger Knublauch, James A. Hendler, and Kingsley Idehen, "SPIN - Overview and Motivation," W3C, Member Submission, Feb. 2011.
- [36] D. Morey, M. T. Maybury, and B. M. Thuraisingham, *Knowledge management: classic and contemporary works*. Cambridge, Mass.: MIT Press, 2002.
- [37] P. Hayes, "RDF Semantics," World Wide Web Consortium, Feb. 2004.
- [38] D. Brickley and R. V. Guha, Eds., *RDF Schema 1.1*. W3C Recommendation, 2014.
- [39] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, "OWL 2 Web Ontology Language Primer," World Wide Web Consortium, W3C Recommendation, Oct. 2009.
- [40] H. Boley, G. Hallmark, M. Kifer, A. Paschke, A. Polleres, and D. Reynolds, Eds., *RIF Core Dialect (Second Edition)*. W3C Recommendation, 2013.
- [41] I. Díaz, J. Llorens, G. Genova, and J. M. Fuentes, "Generating domain representations using a relationship model," *Inf. Syst.*, vol. 30, no. 1, pp. 1–19, Mar. 2005.
- [42] J. M. Alvarez-Rodríguez, J. Llorens, M. Alejandres, and J. M. Fuentes, "OSLC-KM: A knowledge management specification for OSLC-based resources," *INCOSE Int. Symp.*, vol. 25, no. 1, pp. 16–34, Oct. 2015.

Annex I

Feature	RDF [37]	RDFS [38, p. 1]	OWL [39]	RIF Core [40]	SRL [41]
Full Name	Resource Description Framework	Resource Description Framework Scheme	Ontology Web Language	Rule Interchange Format	System Representation Language
First Version	1.0 (February 2004)	1.0 (February 2004)	1.0 (February 2004)	First edition (December 2012)	v1 (January 2004)
Last version	1.1 (February 2014)	1.1 (February 2014)	2.0 (December 2012)	Second edition (February 2013)	v14 (January 2015)
Designed for	Representation of logical statements	Data modeling vocabulary for RDF data	Formal ontology design	Definition of Horn rules	Representation of relationships between knowledge items
Target use	Data exchange of facts, rules and ontologies	Data model	Ontology creation	Rule interchange	Universal knowledge representation and re-use
Data model	Directed graph	Directed graph	Directed graph	Object Model	Undirected (property) graph
Underlying semantics	RDF formal semantics	RDFS Semantics	OWL 2. Direct Semantics and RDF-based Semantics	RIF Core Semantics	Explicit metamodel
Expressivity	Simple RDF triples (s, p, o) to represent binary relationships.	Classes (sub and super classes) and Properties (domain and ranges)	OWL 1.1: <ul style="list-style-type: none"> • DL (Description Logic), • Lite, • Full OWL 2.0: <ul style="list-style-type: none"> • EL (Expressions Language) • QL (Query Language) • RL (Rule Language) 	<ul style="list-style-type: none"> • RIF-Core (Core Dialect) • RIF-BLD (Basic Logic Dialect) • RIF-PRD (Production Rule Dialect) • RIF-FLD (Framework for Logic Dialects) • RIF-OWL 2 RL and RIF RDF • RIF XML 	Any kind of relationship (SVP). <ul style="list-style-type: none"> • N-ary relationships. • Non logic formalism. • Knowledge containers. (reification)
Validation	RDF Data Shapes: <ul style="list-style-type: none"> • OSLC Resource Shapes, SHACL (Shapes Constraint Language) • SheX (Shape Expressions) • SPIN (SPARQL Inferencing Notation) and SPARQL Rules 	Semantic reasoning + see RDF	Semantic reasoning + see RDF	Metamodel conformity	Metamodel conformity
Inference	Not at graph level.	Yes, but restricted to type inference and super/sub classes and properties	Yes, depending on the underlying logic formalism: First Order Logic, F-Logic, DL, etc.	Yes	Not at graph level.
Identifiers	URIs (HTTP URIs if Linked Data). Unique Name Assumption (UNA).	See RDF	See RDF	Internal IDs and UNA.	Internal IDs and UNA.
Access protocol	HTTP-based (REST resources)	See RDF	See RDF	See RDF and native APIs	Native API
Query language	SPARQL and RDQL	See RDF	SWRL	XPATH (if XML is used as serialization format)	RSHP query language
Storage	RDF repository (native RDF repositories, graph-based databases, and wrappers on top of existing relational databases)	See RDF	See RDF	Native API	SQL or NonSQL database
Formats (syntax)	RDF/XML, JSON, Turtle, N3, Manchester	See RDF	See RDF	XML	RDF/XML, ISO 25964-“The international standard for thesauri and interoperability with other vocabularies”, etc.
Visualization	RDF visualization libraries such as Allegro graph or RDFgravity and other general-purpose graph visualization frameworks Graphviz, Touchgraph, Gephi, Cytoscape, D3.js.	See RDF	See RDF	Native Rule IDEs	RSHP visualization language and the aforementioned general-purpose graph visualization frameworks.
Application	Integration of databases, applications and services through a common and shared data model.	See RDF	See RDF	Interchange of business rules and connection with existing ontologies	Semantics-based information retrieval using a natural language interface to support other services such as traceability or quality.
Status	W3C recommendation	W3C recommendation	W3C recommendation	W3C recommendation	Industry-oriented
Tools	Protégé, SWOOP or Terminae, TopBraid Composer (ontology editors)	See RDF and RDFS reasoners such as Pellet, Racer or Jess	See RDFS	JRules, Drools or Jess (mainly exporters not importers)	knowledgeMANAGER (a complete suite for knowledge management with RDF import/export capabilities)

Table 2 A comparison among the main approaches for knowledge representation using an underlying semantic network.

Towards an Ontology to Support Decision-making in Hospital Bed Allocation (TSE)

Debora Engelmann, Julia Couto, Vagner Gabriel, Renata Vieira, and Rafael H. Bordini
School of Technology, PUCRS - Pontifical Catholic University of Rio Grande do Sul - Porto Alegre, Brazil
Email: [debora.engelmann, julia.couto, vagner.gabriel]@edu.pucrs.br, [renata.vieira, rafael.bordini]@pucrs.br

Abstract—Using advanced technologies is imperative to support quick decision-making in a hospital, where people work with complex and critical processes. An example of a complex and very important task is to make the best decision about in which room and bed a patient should be admitted to hospital, considering the patient needs, characteristics, and available resources. In this case, bad decisions can even compromise patient health. With this study, we aim to facilitate patient-related decisions related to bed allocation, based on an ontology. To do so, we developed an ontology that takes patients' information into account to help health professionals decide where to allocate them. Our main contribution is an ontology, with classes, relationships, individuals, and rules to be used in specific scenarios. Additionally, we exemplify some scenarios in which the rules could be applied.

Index Terms—Ontology, health care, hospital bed allocation.

I. INTRODUCTION

Hospital decision-making is an important and complex task, which demands a great deal of cognitive effort from health professionals [13]. In this way, tools and technologies that facilitate decision making are great alternatives to help health professionals reduce their cognitive load and possible errors caused by fatigue.

Information technology is widely adopted in modern medical practice, especially to support administrative tasks, patient electronic records, and data management [15]. Artificial Intelligence (AI) has also been highlighted in several applications in the medical field, e.g. to predict patient inflow [8], identify high-quality physicians based on big data [24], and estimate surgical time duration [21].

Ontologies provide the basis to establish an explicit formal concept specification in a specific domain, allowing the development of relationships between these concepts and the reuse and integration of domain knowledge [10]. In the hospital domain, ontologies provide information associated with a wealth of knowledge about clinical decisions [3], medication [2], diagnoses [19], medical records [11], and so on.

In this paper, we report the development of an ontology for bed allocation, which we developed for the Portuguese language. Our ontology is composed of classes, object properties, and individuals, enabling us to present real scenarios that can be tested using its rules. The main function of our ontology is to assist decision making about the place where patients will be allocated when being admitted to hospital, according to the patient's records, characteristics, and the bed allocation rules used in the particular hospital.

II. THEORETICAL BACKGROUND

The development, dissemination, and use of common communication standards, vocabularies, and ontologies are important for the development of health systems [13]. Therefore, in this section we explore some of the main concepts related to our work, such as hospital bed allocation and ontologies.

A. Hospital bed allocation

Hospital bed management is an important part of operational capacity planning and control, and it involves the efficient use of resources [17]. Each hospital has its own rules, but there are some factors that must be considered when choosing a bed for a patient, such as age, patient condition, and gender.

Effective management of beds has always been a challenge for managers, given that hospital settings are highly dynamic and uncertain. It is necessary to accommodate both scheduled and emergency patients, requiring multiple expertise in a wide range of hospital departments with various different constraints [4].

Furthermore, in a hospital there are many other resources that need to be allocated and many decisions that always need to be made as quickly as possible. An ontology-based decision support system can be useful to quickly make the right decisions, even helping save lives.

B. Ontology

An ontology is an explicit and formal specification of a shared conceptualisation made up of concepts or classes, relationships, instances, attributes, axioms, restrictions, rules, and events [20]. An ontology communicates what kinds of things exist and how they are related to each other [22]. A standard for representing ontologies that is widely used both in academia and industry is the OWL (Ontology Web Language), is a language for representing ontologies that is based on formal logic, a discipline that evolved from philosophy and mathematics [22].

For instance, Protégé¹ is an open source tool that implements OWL. A Protégé ontology consists of classes, properties, individuals, and rules. Classes are concepts in the domain of discourse. Object properties are slots that describe properties or attributes of classes. Individuals are class instances, and rules are axioms specifying additional constraints. The ontology and its class individual instances with specific values comprise a knowledge base on Protégé [14].

III. BED ALLOCATION ONTOLOGY

The inspiring approach [5] was used in the development of the ontology containing 95 classes, 85 object properties, and 78 individuals. We created the original version in Portuguese, but we translated the terms here for consistency with the study report. Next, we explain its components, and present and exemplify the rules we created based on the hospital bed management context. Due to space constraints, we are not able to present here all the details, properties, rules and ontology visual representation, so we provide it in a repository at GitHub: <https://github.com/DeborahEngelmann/Hospital-Bed-Allocation-Ontology>.

A. Classes

In this section, we detail the classes we created. Note that all the concepts described here refer to the scenario created by the researchers, and they can have different meanings in different contexts.

Attendance: The term *attendance* is used to refer to the whole period the patient attended any activity in a hospital, either ambulatorial or hospitalisation. *Ambulatorial* is the assistance that occurs via prior scheduling or by emergency need. It has *Elective* (when it is scheduled) and *Emergency* sub-classes (when it occurs without scheduling, because of an urgent need of the patient). *Hospitalisation* occurs when the patient needs to stay in hospital for more than one day, occupying a *Hospital_Bed*.

Risk_Classification: A risk category assigned to patients when they start being cared for, widely used in the emergency sector to prioritise patients in worst health conditions. We describe it based on Manchester protocol, defined by Mackway-Jones, Marsden, and Windle [9]. Sub-classes range from immediate to non urgent. A patient that has risk of death is classified as *Very_urgent*. Patients with immediate risk of limb loss or loss of organ function are classified as *Very_urgent*. Patients with conditions that can worsen if not helped soon are *Urgent*. Patients with low risk of health damage are *Standard*, and the ones without any immediate risk of health damage are *Non_urgent*.

Temporal_concept: Concepts related to the timing of events. It includes the following sub-classes: Now, Year, Date, Day, Today, Hour, Time interval, Month, and Week.

Document: It refers to the documents generated during or after a patient's attendance. It includes diagnosis, report, prescription, and medical records. (1) *Diagnosis*: made by a doctor, it determines the disease nature and cause, based on the patient history, symptoms, examination, etc. (2) *Report*: made by a specialist doctor, it usually contains the analysis of exams, such as radiology, laboratory, etc. (3) *Prescription*: made by a doctor, it includes drugs and treatments recommended to the patient. (4) *Medical_records*: it includes all the data related to the patient that can be accessed and stored by the hospital.

Disease: Biological alteration of a person's health state, manifested by a set of symptoms.

Speciality: Represents the medical specialisation or expertise that the doctor possesses or that the patient needs.

State: it represents patient conditions, and has 5 sub-classes: Coma, In treatment, Stable, Severe, and Vegetative.

Situation: it represents hospital bed conditions, and has 5 sub-classes: Blocked, Clean, Free, Occupied, and Dirty.

Local: places inside a hospital. Sub-classes are: Corridor, Pharmacy, Bed, Room, Reception, and *Hospitalisation_Unit*. The *hospitalisation_Unit* also has sub-classes named: *Speciality_Unit*, Nursery, Pediatrics, Intensive_Care_Unit, and *Special_Care_Unit*.

Medication: the drugs stored on the Pharmacy, which are meant to treat the patients.

Furniture: All the furniture that belongs to the hospital. For this study, we describe just two sub-classes: Bed and Stretcher.

Person: People who belong to the hospital ecosystem. Sub-classes are: *Companion*, *Man*, *Woman*, *Patient*, and *Employee*. *Employee* has the following sub-classes: *Administration*, *Cleaner*, *Receptionist*, *Security_Guard*, and *Health_Professional*. The last one comprehends *Nurse*, *Nursing_Technician*, and *Doctor*. *Doctor* also have sub-classes named: *Generalist*, *Resident*, and *Specialist*. *Specialist* has the following sub-classes: *Cardiologist*, *Dermatologist*, *Neurologist*, *Oncologist*, *Pediatrician*, *Pneumologist*, *Radiologist*, and *Traumatologist*.

Restriction: rules to restrict bed allocation. (1) *Routing*: Origin of the patient, for example if he came from the emergency or is an elective patient. (2) *Age*: person's age group, which can be adult, teenager, or child. (3) *Gender*: male or female. (4) *Isolation*: Refers to the cases where the patient cannot be in the a room with other patients. (5) *Puerperal*: Women who just gave birth. (6) *Length_Of_Stay*: Predicted time of patient stay in the hospital, can be turn-fast or long-stay. (7) *Hospital_Care*: Hospital care the patient needs, can be surgical or clinical. (8) *Type_Of_Care*: Type of care that the patient needs, can be minimal, semi-intensive or intensive.

Symptom: Signs to which the patient refers when talking about his illness (pain, fever, etc.).

Treatment: Set of instructions of procedures that the doctor recommends for the patient undergo.

B. Object Properties

We created 85 relationships between the classes, and some of them presented in Table I. The full list is available at GitHub. We have not included all possible relationships between classes, only those we consider interesting for this domain, so we could see clearly how the classes relate to each other, to help us test the rules.

C. Individuals

We instantiated 78 individuals, so we could use the reasoner to test our rules. Individuals we created include Patient, Room, Hospital_Bed, Symptoms, and so on. To create them, we used names such as Patient1, 100, 100A, Headache, and so forth.

D. Rules

Our ontology aims to help decision making about the beds where patients can be allocated according to the bed

TABLE I
BED ALLOCATION ONTOLOGY – OBJECT PROPERTIES

Domain	Object Property	Range	Inverse of
Companion	accompanies	Patient	is-accompanied-by
Attendance	happens-in	Temporal_concept	
Nurse	allocates	Hospital_Bed	is-allocated-by
Health_Profess.	analyses	Document	is-analysed-by
Patient	presents-one	Disease	
Employee	attend	Patient	is-attended-by
Doctor	attend-the-speciality	Speciality	
Health_Profes.	evaluates	Patient	is-evaluated-by
Patient	consumes	Medication	is-consumed-by
Doctor	discharges	Patient	is-discharged-by-the
Patient	vacates-one	Hospital_Bed	is-vacated-by
Doctor	diagnoses-one	Patient	is-diagnosed-by

Domain	Object Property	Range	Inverse of
Hospital_Bed	bed-is-of-the-age-group	Age	
Hospital_Bed	bed-is-the-attendance	Hospital_Care	
Hospital_Bed	bed-is-care	Type_Of_Care	
Hospital_Bed	bed-is-routing	Routing	
Hospital_Bed	bed-is-stay	Length_Of_Stay	
Hospital_Bed	bed-is-of-the-gender	Gender	
Hospital_Bed	bed-is-own-one	Puerperal	
Hospital_Bed	bedroom-is-speciality	Situation	
Bedroom	bedroom-is-the-attendance	Speciality	
Bedroom	bedroom-is-care	Hospital_Care	
Bedroom	bedroom-is-routing	Type_Of_Care	
Bedroom	bedroom-is-routing	Routing	

allocation constraints. Thus, we establish rules that propagate information about restrictions by registered individuals. We are aware that a lot of rules can be created to help decision making related to bed allocation in hospitals. We present a sample of the ones we created for our ontology in Table II.

1) *First scenario*: For some hospitals, there are a number of bedrooms that have no preferential gender. It means one can allocate male or female patients to those bedrooms, but not both. Let us say there is a bedroom with two empty beds, and then a man is allocated to one of those beds. Thereafter, while there is a man in this bedroom, only another man can occupy the other bed. Nevertheless, as soon as the bedroom is empty, women can occupy it too, according to the priority in the queue of patients to be hospitalised. In this scenario, our rules work as follows. If a male patient is placed in a bed, RULE 1 infers that the patient is male, RULE 23 infers that the bed has become male only, RULE 29 infers that if a bed of the bedroom is for male patients, then the bedroom must also be male only, and finally, RULE 19 infers that if a bedroom is male only, all the beds in that room are too.

2) *Second scenario*: Patient disease and condition also influences bed allocation. For instance, when a woman is suspected to have a disease transmitted by air and highly contaminating, she must be isolated, having her own bedroom, and be attended by a Pneumologist doctor. In this case, when allocating this patient in a bed, RULE 21 infers that if the patient is of the Pneumology speciality then the bed where she was allocated is also of that speciality. RULE 9 infers that the bedroom where this bed is also is of the Pulmonology speciality. Finally, RULE 8 infers that if there are other beds in that bedroom, they all have that same speciality. In addition, because it is a patient in isolation, rule 20 infers that if the patient is in isolation, so is the bed. RULE 2 infers that if the bed is for isolation then the bedroom should also be. And

finally, RULE 27 infers that all beds in the bedroom should be of the same type of isolation.

3) *Third scenario*: Another important restriction related to bed allocation is age group. When a child has to be hospitalised, we can only allocate them in the same bedroom with people up to 12 years old. The rules we can apply in this case are: RULE 24 that infers that if the patient in a bed is of a certain age group then the bed is also; RULE 16 infers that the bedroom is the same age group as the bed that has a person allocated; and RULE 18 that infers that all other beds in the bedroom have the same age group.

4) *Fourth scenario*: When a healthy baby is born, its mother must be allocated in a bed in the Maternity Unit, because she needs a baby crib next to her bed. In this case, RULE 12 infers that if the patient is of a puerperal type, then the bed is of the same type. RULE 7 infers that the bedroom to which the bed belongs will be of the same puerperal type, and RULE 28 infers that in this case, the other beds in that bedroom must only be allocated women who just gave birth to a healthy child. In addition to these scenarios, we create rules with this same type of reasoning for all other constraints.

IV. RELATED WORK

A considerable number of papers studied ontologies on hospital domains. In this Section, we summarise some of them. Rubin [18] presents Radlex, a standard terminology widely used in the radiology field. Mhiri and Despres [12] use ontologies as indexing support to generate more accurate clinical reports. Kataria et al. developed HIWO [6], a formal description of an intelligent hospital domain, based on a ontology. The Nurse Call System (oCS) [15], developed by Ongenae et al., uses an ontology to manage context information about patient profiles and team members. An Ontology for Hospital Scenarios (OntHoS), created by Becker et al. [1], aims to establish a basis for a description of these

TABLE II
RULES OF THE ONTOLOGY

Rules

1. *Patient(?X), Man(?X) → is - of - the - gender(?X, Male)*
2. *Hospital_Bed(?Y), Bedroom(?Z), is - in(?Y, ?Z), bed - is - isolation(?Y, ?I) → bedroom - is - isolation(?Z, ?I)*
7. *Hospital_Bed(?Y), Bedroom(?Z), is - in(?Y, ?Z), bed - is - puerperal(?Y, ?Q) → bedroom - is - puerperal(?Z, ?Q)*
8. *Hospital_Bed(?Y), Bedroom(?Z), is - in(?Y, ?Z), bedroom - is - speciality(?Z, ?S) → bed - is - speciality(?Y, ?S)*
9. *Hospital_Bed(?Y), Bedroom(?Z), is - in(?Y, ?Z), bed - is - speciality(?Y, ?S) → bedroom - is - speciality(?Z, ?S)*
12. *Patient(?X), is - puerperal(?X, ?Q), Hospital_Bed(?Y), occupy - one(?X, ?Y) → bed - is - puerperal(?Y, ?Q)*
16. *Hospital_Bed(?Y), Bedroom(?Z), is - in(?Y, ?Z), bed - is - of - the - age - group(?Y, ?G) → bedroom - is - of - the - age - group(?Z, ?G)*
18. *Hospital_Bed(?Y), Bedroom(?Z), is - in(?Y, ?Z), bedroom - is - of - the - age - group(?Z, ?G) → bed - is - of - the - age - group(?Y, ?G)*
19. *Hospital_Bed(?Y), Bedroom(?Z), is - in(?Y, ?Z), bedroom - is - of - the - gender(?Z, ?H) → bed - is - of - the - gender(?Y, ?H)*
21. *Patient(?X), is - speciality(?X, ?S), Hospital_Bed(?Y), occupy - one(?X, ?Y) → bed - is - speciality(?Y, ?S)*
23. *Patient(?X), is - of - the - gender(?X, ?H), Hospital_Bed(?Y), occupy - one(?X, ?Y) → bed - is - of - the - gender(?Y, ?H)*
24. *Patient(?X), is - of - the - age - group(?X, ?G), Hospital_Bed(?Y), occupy - one(?X, ?Y) → bed - is - of - the - age - group(?Y, ?G)*
27. *Hospital_Bed(?Y), Bedroom(?Z), is - in(?Y, ?Z), bedroom - is - isolation(?Z, ?I) → bed - is - isolation(?Y, ?I)*
28. *Hospital_Bed(?Y), Bedroom(?Z), is - in(?Y, ?Z), bedroom - is - puerperal(?Z, ?Q) → bed - is - puerperal(?Y, ?Q)*
29. *Hospital_Bed(?Y), Bedroom(?Z), is - in(?Y, ?Z), bed - is - of - the - gender(?Y, ?H) → bedroom - is - of - the - gender(?Z, ?H)*

scenarios and facilitate their interoperability. We also found a UML-based ontology to describe hospital information system architectures [23], an ontology for intelligent assistants in patient management based on rules [16], and an ontology-based health context information model to implement omnipresent environments [7]. Different from our work, none of them focuses on bed allocation.

V. CONCLUSIONS

In this paper we presented the concepts, properties, and rules we developed for an ontology that can be used to help decision making about the place where the patients will be allocated when hospitalised, according to predefined bed allocation rules. We also present four different examples of scenarios in which the ontology could be used. Our ontology has not been used yet in a real scenario, although it is freely available online and we hope it can motivate people to use it. The full version of our ontology has 95 classes, 85 object properties, and 78 individuals. As future work we plan to evaluate our ontology with domain specialists. Our ontology can also be extended and explored in other directions such as the definition of the number of patients per nurse, distribution and classification of the patients in beds, among other possibilities.

REFERENCES

- [1] M. Becker *et al.*, *OntHoS - an Ontology for Hospital Scenarios*. Birkhäuser Verlag, 2003, pp. 87–103.
- [2] J.-J. Chen *et al.*, “Applying ontology techniques to develop a medication history search and alert system in department of nuclear medicine,” *Journal of Medical Systems*, vol. 36, no. 2, pp. 737–746, Apr 2012.
- [3] B. Cánovas-Segura *et al.*, “A lightweight acquisition of expert rules for interoperable clinical decision support systems,” *Knowledge-Based Systems*, vol. 167, pp. 98 – 113, 2019.
- [4] M. d. S. Grüber *et al.*, “A hospital bed allocation hybrid model based on situation awareness,” *CIN*, vol. 36, no. 5, pp. 249–255, May 2018.
- [5] C. W. Holsapple *et al.*, “A collaborative approach to ontology design,” *Commun. ACM*, vol. 45, no. 2, pp. 42–47, Feb 2002.
- [6] P. Kataria *et al.*, “Implementation of ontology for intelligent hospital wards,” in *HICSS*. IEEE Computer Society, 2008, p. 253.
- [7] J. Kim *et al.*, “Ontology-based healthcare context information model to implement ubiquitous environment,” *Multimedia Tools and Applications*, vol. 71, no. 2, pp. 873–888, Jul 2014.
- [8] D. A. Kottalanka Srikanth, “An efficient patient inflow prediction model for hospital resource management,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 7, no. 3, pp. 809–817, 2017.
- [9] K. Mackway-Jones *et al.*, *Emergency triage: Manchester triage group*. John Wiley & Sons, 2013.
- [10] H. Martin *et al.*, “Medical ontologies for machine learning and decision support,” mar 2011, uS Patent 7,899,764.
- [11] R. Messaoudi *et al.*, “An ontological model for analyzing liver cancer medical reports,” in *Information Systems*. M. Themistocleous *et al.*, Eds. Cham: Springer International Publishing, 2019, pp. 369–382.
- [12] S. Mhiri *et al.*, “Ontology usability via a visualization tool for the semantic indexing of medical reports (dicom sr),” in *USAB*. Springer, 2007, pp. 409–414.
- [13] J. Nealon *et al.*, “Agent-based applications in health care,” in *Applications of software agent technology in the health care domain*. Springer, 2003, pp. 3–18.
- [14] N. F. Noy *et al.*, “The knowledge model of protégé-2000: Combining interoperability and flexibility,” in *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, R. Dieng *et al.*, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 17–32.
- [15] F. Ongena *et al.*, “An ontology-based nurse call management system (oncs) with probabilistic priority assessment,” *BMC Health Services Research*, vol. 11, no. 1, p. 26, Feb. 2011.
- [16] V. L. Payne *et al.*, “Hospital care watch (hcw): an ontology and rule-based intelligent patient management assistant,” in *CBMS*, June 2005, pp. 479–484.
- [17] N. C. Proudlove *et al.*, “Can good bed management solve the overcrowding in accident and emergency departments?” *Emergency Medicine Journal*, vol. 20, no. 2, pp. 149–155, 2003.
- [18] D. L. Rubin, “Creating and curating a terminology for radiology: ontology modeling and analysis,” *Journal of digital imaging*, vol. 21, no. 4, pp. 355–362, 2008.
- [19] L. Subirats *et al.*, *Personalization of Ontologies Visualization: Use Case of Diabetes*. Cham: Springer International Publishing, 2019, pp. 3–24.
- [20] M. Tovar *et al.*, “A metric for the evaluation of restricted domain ontologies,” *Computación y Sistemas*, vol. 22, no. 1, pp. 147–162, Mar 2018.
- [21] J. P. Tuwatananurak *et al.*, “Machine learning can improve estimation of surgical case duration: A pilot study,” *Journal of Medical Systems*, vol. 43, no. 3, p. 44, Jan 2019.
- [22] M. Uschold, “Demystifying owl for the enterprise,” *Synthesis Lectures on Semantic Web: Theory and Technology*, vol. 8, no. 1, pp. i–237, May 2018.
- [23] A. Winter *et al.*, “A uml-based ontology for describing hospital information system architectures,” *Studies in health technology and informatics*, vol. 84, no. Pt 1, p. 778–782, 2001.
- [24] Y. Ye *et al.*, “A hybrid it framework for identifying high-quality physicians using big data analytics,” *International Journal of Information Management*, vol. 47, pp. 65 – 75, 2019.

A Software System is Greater than its Modules' Sum: Providers & Consumers' Modularity Matrix (TSE)

Iaakov Exman and Harel Wallach
Software Engineering Department
The Jerusalem College of Engineering – JCE - Azrieli
Jerusalem, Israel
iaakov@jce.ac.il, harel.wallach@gmail.com

Abstract— Modularity Matrices and their Laplacians enable finding software system modules by a rigorous algebraic procedure. However, Modularity Matrices have up to now focused mainly on structors as providers of functionals. This paper takes a broader view at the software system as a whole. The Software System Modularity Matrix, besides displaying *provider* relationships, also describes which structors *consume* functionals provided by other structors. This broader view improves software system design in two ways. First, consumer relationships set realistic expectations for consumer numbers and roles. Second, the Software System Modularity Matrix generates standard design criteria for interacting providers and consumers. This standard System matrix obeys linear independence of its constituent vectors, and block-diagonality of its recognizable modules. The novelty consists of modules being composed into a whole working Software System by means of a limited number of consumers playing the role of module *connectors*. Modules and their connectors are formally obtained by the same spectral method applied to the respective Laplacian, which obtained provider matrices. This is illustrated by case studies.

Keywords: Linear Software Models; Spectral Software Design; Modularity Matrix; Laplacian Matrix; Providers; Consumers; Modules; Connectors; Linear Independence; Block-Diagonality.

I. INTRODUCTION

Linear Software Models represent each abstraction level of a given Software System by means of a Modularity Matrix [8], [10] or its corresponding Laplacian Matrix [13], [14]. These models enable rigorous software design of any given system:

- **Standard Matrix** – is defined as an exact criterion to compare different proposed designs;
- **Matrices highlight the need of redesign** – pinpointing eventual coupling problem locations.

A Modularity Matrix is made of column *structors* – a generalization of classes in object-oriented programming languages – and row *functionals* – a generalization of class methods. A matrix element is 1-valued if its structor provides the respective functional. Otherwise, the element is zero-valued.

The meaning of a structor as a *provider* of a functional is e.g. a class containing the declaration/definition of a method, usable by other classes. Another structor using such a functional is called a *consumer* (e.g. a class calls a method of another class).

This paper's goal is to propose and deal with the broader sense of Modularity Matrix, and its corresponding Laplacian, displaying not only providers, but also matrix elements standing for consumers. This extended Modularity Matrix is a more complete representation of Software from the system perspective and is of practical interest for software design.

This Introduction concisely reviews concepts of Modularity and Laplacian matrices.

A. Software Modularity

A central problem to be solved by software engineering is the hierarchical composition of a software system from sub-systems, down to software architecture units, typically classes, considered indivisible by the designer.

Solving the software system composition problem involves software modularity. Applying Linear Software Models, one designs one or more Modularity Matrices, obtaining modules by spectral methods. One then compares their quality with a square and block-diagonal standard matrix, resolving eventual coupling problems, highlighted by the matrices.

A simple example of a schematic Modularity Matrix with providers only is shown in Fig. 1. It has five structors and five functionals. It displays three modules, the blocks along the diagonal. It is a standard Modularity Matrix as it does not have any outlier, a 1-valued matrix element outside the modules.

B. Modularity Matrices and their Laplacians

A Laplacian Matrix is easily generated from a Modularity Matrix in two steps:

- **Extract a bipartite graph** – with a structors' vertex set and a functionals' vertex set having edges corresponding to 1-valued matrix elements of the Modularity Matrix;
- **Generate the Laplacian Matrix** – from the bipartite graph, according to equation (1):

$$L = D - A \quad (1)$$

where L is the Laplacian matrix, D is the Degree matrix of the graph vertices and A is the Adjacency matrix of vertex pairs.

	S1	S2	S3	S4	S5
F1	1	1			
F2	0	1	All zeros		
F3			1		
F4	All zeros			1	1
F5				0	1

Figure 1. Schematic Providers Modularity Matrix – It has 5 structors (S1 to S5), 5 functionals (F1 to F5) and three modules, the (blue) blocks along the diagonal: upper-left and lower-right with 2*2 size and middle block of size 1*1. It is a standard matrix as it does not have outliers (1-valued elements outside modules). (All figures are in color online).

A bipartite graph, obtained from the Modularity Matrix in Fig. 1, is shown in Fig. 2.

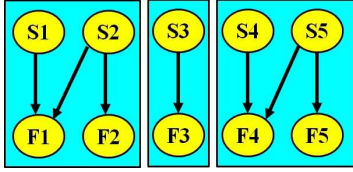


Figure 2. Bipartite Graph from Modularity Matrix in Fig. 1 – It has two vertex sets: the upper set of structors (S1 to S5), and the lower set of functionals (F1 to F5). A bipartite graph only has edges linking vertices in different sets. Arrows pointing down mean that structors provide functionals. The (blue) rectangles separate vertices belonging to given connected components (the modules).

A schematic Laplacian Matrix, generated from the bipartite graph in Fig. 2, is shown in Fig. 3.

	F1	F2	F3	F4	F5	S1	S2	S3	S4	S5
F1	2					-1	-1			
F2		1				0	-1			
F3			1					-1		
F4				2					-1	-1
F5					1				0	-1
S1	-1	0				1				
S2	-1	-1					2			
S3			-1					1		
S4				-1	0				1	
S5				-1	-1					2

Figure 3. Schematic Providers Laplacian Matrix – This Laplacian is generated from the bipartite graph in Fig. 2. By equation (1) its diagonal is D the Degree matrix (in green) showing the degrees of each vertex of the Bipartite graph. The upper-right quadrant (and its reflection in the lower-left quadrant) is the negative of A the graph Adjacency matrix, which is identical to the Modularity Matrix.

C. Paper Organization

The rest of the paper is organized as follows. Section II mentions related work. Section III introduces Consumer Matrices to be included in the whole System Matrices, which is done in Section IV. Section V illustrates the provider and consumer matrices by means of a server case study. Section VI concludes the paper with a discussion.

II. RELATED WORK

Here one finds a concise review of the extensive literature about Modularity approaches, by spectral and other methods. Also shortly reviewed are some references to consumers.

A. Linear Software Models

Linear Software Models have been developed by Exman and collaborators (e.g. [8], [9]) as a rigorous theory to solve the hierarchical software system composition problem from sub-systems. Linear Software Models are based on linear algebra operations and theorems. One assumes that all structors should be mutually linearly independent and also all functionals are linearly independent, an assumption motivated by minimization of the number of structors and functionals needed to build the system. Given this assumption, a linear algebra theorem demands that the Modularity Matrix be square. This is not a trivial result for software systems; it demands some effort to understand the theorem's rationale and implications.

Moreover, if sub-sets of structors/functionals are disjoint to other sub-sets, a second theorem states that these sub-sets can be rearranged into a block-diagonal matrix. These diagonal blocks are recognized as the modules in that software system level (for detailed proofs and examples see the work by Exman [10] and references therein).

A given software system modularization may display undesirable provider outliers coupling between modules. A procedure to compare different designs of the same software system, and to improve design is given by spectral methods as described in [11]. The Perron-Frobenius theorem (see e.g. Gantmacher [17]) is central for the Modularity Matrix theory.

Exman and Sakhnini [13], [14] have shown how to generate a Laplacian Matrix from the Modularity Matrix. The Laplacian matrix obtains the same modules as the Modularity Matrix, by similar spectral methods. The Fiedler theorem [1], [15] is central for the Laplacian theory. The so-called Fiedler eigenvector fits the lowest non-zero eigenvalue of the Laplacian Matrix. It allows locating outliers and splitting of too sparse software modules.

B. Alternative Approaches to Modularity

There exist a variety of techniques applying matrices for modularity analysis. For instance, Baldwin and Clark describe a Design Structure Matrix (DSM) in their "Design Rules" book [2]. DSM has been applied to many systems, including software engineering, see e.g. Cai and Sullivan [5].

Conceptual lattices, another algebraic structure relevant to software design, were introduced by Wille in 1982 [21] as part of Formal Concept Analysis (FCA). They have been used for

software system design e.g. by Siff and Reps [19] and by Exman and Speicher [12].

Alternative clustering techniques to obtain software modules are found e.g. in Shtern and Tzerpos [18].

C. Theoretical Approaches to Consumers

There have been modelling systems in the literature representing provider and consumer interactions. Yau and Caglayan [22] use Petri Nets to design distributed software systems. One of their examples is a producer-consumer system.

Clark et al. [6] describe experiences with PEPA (Performance Evaluation Process Algebra) modelling tools. In particular they refer to Producer-Consumer relations.

Browning [4] suggests that system modelers often build two DSM matrices, one for information supplier and another for the consumer, similar to our providers/consumers pairs of matrices.

III. THE NATURE OF CONSUMER MATRICES

Consumer matrices display Structors and Functionals consumed by the referred Structors. This section describes assumptions needed to generate Consumer matrices.

A. Consumer Matrices Shape and Size

Consumer matrices are not by themselves the aim of this paper. The goal of consumer matrices, jointly with provider matrices, is a more complete description of a software system, showing the interactions between the given sets of Structors and Functionals, from a system perspective.

Given this goal, we assume that Structors and Functionals of the consumer matrices are identical to those of the provider matrices. Thus, a consumer matrix fitting a standard providers' matrix is also square. We emphasize that the reason for consumer matrices being square is essentially different from the provider matrices. As stated in section II standard provider matrices are square by algebraic considerations. Consumer matrices are square just to enable unification of providers and consumers into a single system matrix, as described below.

We often refer to sub-systems, to allow for the possibility that consumer matrices leave outside, e.g. service functionality, obtained from external libraries. Alternatively, functionals provided by our sub-system may be consumed by other sub-systems, not included in our provider/consumer matrices.

Modularity and Laplacian matrices are tools to solve software design problems resulting from coupling interactions between different architectural units – structors and their functionals – in a given hierarchical level. Functionals provided and consumed by the same Structor do not appear in either of the provider/consumer matrices, as these are not interactions between different structors at that level.

B. Theoretical Properties of Consumer Matrices

Given the above assumptions on Consumer Matrices, we state easily verifiable theoretical properties.

Property 1 – Complementarity to Provider Matrices.

Since Consumer Matrices have exactly the same Structors and Functionals as the Provider Matrices, and the matrices do

not display Functionals provided and consumed by the same Structor, consumer Matrices are complementary to Provider matrices of the same sub-system. In other words, there is no overlap of non-zero matrix elements of the consumer matrix with non-zero matrix elements of the provider matrix.

Property 2 – Consumer Matrices may have empty (totally zero-valued) columns or rows, while Provider matrices cannot.

This may happen since the Sub-system Under Design (SUD) may interact with other external sub-systems or libraries not represented in the matrices of the SUD. For example, if an SUD Structor in the Provider matrix provides a Functional consumed only by external sub-systems, the respective SUD Consumer matrix will have an **empty row** corresponding to the Functional consumed externally. Another example, if an SUD Structor appearing in the Provider matrix does not consume any Functional, the respective SUD Consumer matrix will have an **empty column** corresponding to the referred Structor. Provider matrices cannot have empty columns or empty rows, since they display only Structors actually providing Functionals.

Property 3 – Consumer Matrices per se generally neither display linear independence of their Structors/Functionals, nor have block-diagonal modules, in contrast with Provider matrices.

This happens, since besides the empty columns/rows already mentioned in the previous property, it may be that a single given Structor consumes several Functionals originating identical rows. In other words, the rank of a Consumer Matrix is generally less than its size would permit. This does not occur with Provider matrices as already mentioned in section II.

C. An Example of Consumer Matrix

Now we reveal that the provider modularity matrix in Fig. 1 refers to the Command Design Pattern code in CSharp found in [7]. We use the same Structors and Functionals to show the respective Consumer matrix (in Fig. 4). The Command Design Pattern serves as an introductory running example, and as a first case study, in this and in the next section.

Structors →		ICommand	Concrete File Command	File Operation Invoker	IFile Operator	Concrete File Operator
Functionals ↓		S1	S2	S3	S4	S5
Execute	F1			1		
Create Command	F2					
InvokeAll	F3					
File operations	F4		1			
Create Operator	F5					

Figure 4. Command Design Pattern, Consumers only Modularity Matrix – This consumers matrix fits the Providers Modularity Matrix in Fig. 1. Both matrices have the same Structors (S1,...,S5) and the same Functionals (F1,...,F5), and both comply with the Properties enumerated in sub-section B. The consumers matrix has just two 1-valued matrix elements (green hatched background), respectively (S3, F1) and (S2,F4) and is much sparser than the providers matrix.

IV. SYSTEM MODULARITY MATRICES: PROVIDERS AND CONSUMERS

This section finally deals with whole system modularity matrices including providers and consumers. The same algebraic techniques, previously used to identify provider-only matrix modules, are applied for the whole system matrices. This is done here for the Command Design Pattern Laplacian matrix.

A. System Weighted Modularity Matrices

We obtain the System Modularity Matrix by straightforward superposition of the provider matrix with the consumer matrix in a single overall matrix. This is possible as, by *Property 1* above, there is no overlap between non-zero matrix elements of these two matrices. But simple superposition would imply loss of “direction” information, i.e. whether a Functional is provided or consumed by a given Structor. To avoid this ambiguity one assigns a different weight to each direction: a functional *provided* by a structor is assigned a weight of “2” and a functional *consumed* by a structor is assigned a weight of “1”.

In this context, it is important to state that Fiedler [15] has extended the algebraic connectivity properties of Laplacians to those for weighted edge graphs (see e.g. de Abreu [1]).

A System Weighted Modularity Matrix for the Command Design Pattern is shown in Fig. 5, combining the provider matrix of Fig. 1 with the consumer matrix of Fig. 4.

Structors → Functionals ↓		ICommand	Concrete File Command	File Operation Invoker	IFile Operator	Concrete File Operator
		S1	S2	S3	S4	S5
Execute	F1	2	2	1		
Create Command	F2	0	2			
InvokeAll	F3			2		
File operations	F4		1		2	2
Create Operator	F5				0	2

Figure 5. Command Design Pattern, System Weighted Modularity Matrix – This matrix is obtained by superposition of the Consumers Matrix in Fig. 4 with weights of “1”, upon the Providers Matrix (blue modules) in Fig. 1 with weights of “2”, to distinguish the consumers from the providers *direction*.

The weighted bipartite graph obtained from the System Modularity Matrix in Fig. 5, is shown in Fig. 6.

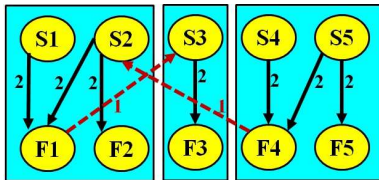


Figure 6. Command Pattern Weighted Bipartite Graph from Modularity Matrix in Fig. 5 – It has two vertex sets: the upper set of structors (S1 to S5), and the lower set of functionals (F1 to F5). Structors providing functionals are shown by (black) arrows pointing down with weight=2. Structors consuming functionals are shown by (red) arrows pointing up with weight=1. The (blue) rectangles denote providers’ connected components (within the providers’ modules). Consumer arrows are *connectors* between providers’ modules.

B. Generation of the Weighted Laplacian

In order to identify the whole system modules including providers and consumers, we obtain from the Weighted Bipartite Graph (Fig. 6), the Weighted Laplacian Matrix in Fig. 7.

C. Connector Discovery from the Weighted Laplacian

As a last step towards the modules of the whole Command Pattern system, including both providers and consumers, we apply the same algebraic spectral method previously used (in [14]) for the providers-only Laplacian. It consists of:

- Calculate eigenvalues and eigenvectors – of the Laplacian Matrix;
- Obtain Modules from eigenvectors – whose eigenvalues are zero-valued;
- Discover Module connectors by splitting modules – using the Fiedler eigenvector.

	F1	F2	F3	F4	F5	S1	S2	S3	S4	S5
F1	5					-2	-2	1		
F2		2				0	-2			
F3			2					-2		
F4				5			1		-2	-2
F5					2				0	-2
S1	-2	0				2				
S2	-2	-2		1			5			
S3	1		-2					3		
S4				-2	0				2	
S5				-2	-2					4

Figure 7. Command Pattern Weighted Laplacian Matrix from bipartite graph in Fig. 6 – It weights (by 2) Laplacian providers in Fig. 3, and adds the consumer elements, with negative weight=1 and a hatched (green) background. Diagonal degrees are changed to guarantee that all rows and columns sum to zero.

Laplacian Matrix Eigenvalues

Eigenvalues are shown in Fig. 8. The only zero-valued eigenvalue is the sixth one: it shows *Modules* in the Laplacian by the fitting eigenvector. The lowest eigenvalue closer to zero is the seventh one and is the *Fiedler* eigenvector, which allows further splitting of the Module.

#1	2	3	4	5	6	7	8	9	10
8.15	7.11	5.00	4.27	4.00	0.00	0.24	0.67	1.35	1.22
Type	→				Modules	Fiedler			

Figure 8. Command Pattern Weighted Laplacian Matrix eigenvalues – these are shown in the middle row of the figure.

Laplacian Matrix Eigenvectors

Eigenvectors' in Fig. 9 fit the Fig. 8 eigenvalues: the 2nd row from the top *Modules* eigenvector has all equal elements, implying one big whole system module; the 3rd row *Fiedler* eigenvector splits the whole system into two modules, by its element signs. Negative signs cluster (F1, F2, F3, S1, S2, S3) into one module and positive signs the vertices (F4, F5, S4, S5) into another module. In this first splitting iteration, the single structor module (F3, S3) seen in Fig. 5 and in Fig. 6, is left inside the 1st module. The 2nd Fiedler vector splitting iteration, in the bottom row of Fig. 9, finally separates the smaller module (F3, S3), obtaining all the three modules in this system.

Laplacian eigenvectors obtain only modules (either directly or by Fiedler vector splitting), as modules are mathematically "connected components" [20] of the graph.

Consumers, as external "connectors", are the remaining positive elements of the Modularity Matrix *by exclusion*, after the modules were directly characterized. For instance, in the 1st splitting iteration the consumer (F1,S3) is left inside the 1st module, while the consumer (F4,S2) is outside both modules. In the 2nd splitting iteration which obtains all modules, also obtains by exclusion both external connectors.

F1	F2	F3	F4	F5	S1	S2	S3	S4	S5
0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32
-0.20	-0.06	-0.47	+0.28	+0.43	-0.23	-0.05	-0.41	+0.32	+0.38
0.12	0.47	-0.64	0.00	0.00	0.17	0.34	-0.47	0.00	0.00

Figure 9. Command Pattern Weighted Laplacian eigenvectors – the top row has vertex indices; the 2nd row from top has all *Modules* vector elements equal 0.32; the 3rd row *Fiedler* different sign elements split the system into two modules 3*3 (negative, blue) and 2*2 (positive, yellow); the bottom 2nd *Fiedler* iteration splits the previous biggest module into 2*2 (positive, green) and 1*1 (negative).

Connectors Discovery by Splitting Modules

The final iteration from the Laplacian eigenvectors is displayed in the System Modularity Matrix, with the referred modules enclosed within dashed rectangles, as seen in Fig. 10.

Structors →		ICommand	Concrete File Command	File Operation Invoker	IFile Operator	Concrete File Operator
Functionals ↓		S1	S2	S3	S4	S5
Execute	F1	2	2	1		
Create Command	F2	0	2			
InvokeAll	F3			2		
File operations	F4		1		2	2
Create Operator	F5				0	2

Figure 10. Command Design Pattern, System Weighted Modularity Matrix with Connectors – This matrix shows the provider modules as the result of the Laplacian eigenvectors, seen as delimited by the dashed (black) rectangles. These are the upper-left and lower-right modules of 2*2 size and the middle 1*1 module. The consumer (F1,S3) links the upper-left and middle modules. The upper-left module is also linked to the lower-right module by the consumer (F4,S2). Iterative splitting by the Fiedler vector obtains these two connectors.

The conclusion from this Command Pattern example is that consumers are *connectors* linking provider modules. This is seen in the bipartite graph (in Fig. 6), and corroborated by the partition by the Laplacian eigenvectors (in Fig. 10).

V. CASE STUDY: AN ASYNCHRONOUS SERVER SYSTEM

This case study is a larger system from the Boost library written in C++, viz. an Asynchronous Echo Server System [3]. The calculation steps were the same as in the Command Pattern example. Here are shown only the important steps' results.

A. Provider and Consumer Modularity Matrices

The Providers Modularity Matrix is strictly diagonal (Fig. 11). The Consumers' Modularity Matrix in Fig. 12 is very sparse and barely understood. Consumption is concentrated in "control" structors, viz. Main, Server and Session.

Structors →		Main	Io-context	Server	Acceptor	Endpoint	Socket	Session	Buffer
Functionals ↓		S1	S2	S3	S4	S5	S6	S7	S8
main	F1	1							
run	F2		1						
server-constructor	F3			1					
async-accept	F4				1				
v4(), port	F5					1			
read, write, is-open, close	F6						1		
start, do-read, do-write	F7							1	
to-stream	F8								1

Figure 11. Asynchronous Server System – Providers-only strictly diagonal Modularity Matrix.

B. Spectral Approach to System Modules

In order to obtain System Modules, we apply the spectral method as done previously with the Command Design Pattern. One superposes the providers and consumers in a single weighted Modularity Matrix, and obtains the bipartite graph. Then one generates its Laplacian Matrix and calculate its eigenvalues and eigenvectors, as shown in Fig. 13 and Fig. 14.

Structors →		Main	Io-context	Server	Acceptor	Endpoint	Socket	Session	Buffer
Functionals ↓		S1	S2	S3	S4	S5	S6	S7	S8
main	F1	1							
run	F2		1						
server-constructor	F3			1					
async-accept	F4				1				
v4(), port	F5					1			
read, write, is-open, close	F6						1		
start, do-read, do-write	F7							1	
to-stream	F8								1

Figure 12. Asynchronous Server System Consumers only Modularity Matrix – This consumers matrix fits the Providers Modularity Matrix in Fig. 11. Both matrices have the same Structors (S1,...,S8) and the same Functionals (F1,...,F8), and both comply with the Properties enumerated in sub-section III B. The non-zero consumers' matrix elements are marked (in green hatched background).

#1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
7.10	6.09	5.73	4.00	4.24	4.38	1.30	0.81	0.00	0.11	0.23	4.56	2.00	0.43	4.56	0.43
Type	→							Module	Fiedler						

Figure 13. Asynchronous Server System Laplacian Eigenvalues – The eigenvalue #9 is the only one zero-valued, implying one *Module* in this system. Eigenvalue #10 is the *Fiedler* vector allowing splitting of this overall module.

The Modules and the Fiedler eigenvectors are shown in Fig. 14. The Fiedler eigenvalue #10 splits the whole system into two modules of 5*5 and 3*3 sizes. The next iteration Fiedler vector (the lowest row) further splits the 5*5 module into two smaller modules of sizes 2*2 and 3*3 sizes.

F1	F2	F3	F4	F5	F6	F7	F8	S1	S2	S3	S4	S5	S6	S7	S8
0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
0.28	0.38	0.04	0.06	0.06	-0.34	-0.24	-0.34	0.29	0.36	0.12	0.05	0.05	-0.32	-0.15	-0.32
-0.25	-0.48	0.17	0.33	0.33	0.00	0.00	0.00	-0.28	-0.43	0.03	0.29	0.29	0.00	0.00	0.00

Figure 14. Asynchronous Server System Laplacian Eigenvectors – From top row to bottom: the 1st row shows the vertices (on yellow); the 2nd row is the eigenvector fitting the single module eigenvalue #9; the 3rd row is the Fiedler #10 eigenvector splitting the 2nd row eigenvector into two modules according to the positive and negative signs; the lowest row is the next iteration Fiedler eigenvector, splitting the larger (green) module in the 3rd row eigenvector into two smaller modules of sizes 2*2 and 3*3 (light blue and orange background).

C. Connector Discovery in the System Matrix

Modules are discovered by looking at the eigenvectors of the Laplacian. These modules are the upper-left (F1,F2,S1,S2), the middle (F3,F4,F5,S3,S4,S5) and the lower-right (F6,F7,F8,S6,S7,S8) as shown in Fig. 15.

External consumers, the “connectors” between the three referred modules, viz. (S1, F3) and (S3, F7) are discovered by exclusion (i.e. outliers), as the remaining positive matrix elements of the System Modularity Matrix outside the three modules (Fig. 15). An interesting observation for this system is the existence of consumers as “internal connectors”, inside the three modules, where each consumer reasonably links a pair of provider structors.

Functionals ↓	Structors →							
	Main	Io-context	Server	Acceptor	Endpoint	Socket	Session	Buffer
	S1	S2	S3	S4	S5	S6	S7	S8
main	F1	2						
run	F2	1	2					
server-constructor	F3	1		2				
async-accept	F4			1	2			
v4(), port	F5			1		2		
read, write, is-open, close	F6						2	1
start, do-read, do-write	F7			1				2
to-stream	F8						1	2

Figure 15. Asynchronous Server System Modules – The three modules enclose the original Provider modules and respective internal connectors. The external connectors are found in the elements (S1,F3) linking the upper-left and middle modules, and (S3,F7) linking the middle and lower-right modules.

VI. DISCUSSION

A. Interpretation of System Matrix Results

A Consumer Matrix by itself is rather perplexing: it is very sparse and not easily interpreted. Consumer Matrices do not obey any apparent algebraic rules such as structors or functionals linear independence, or module block-diagonality, as for Provider Matrix. There are no obvious correctness criteria for Consumer Matrices by themselves.

When one superposes a Consumer Matrix upon its Provider Matrix the picture suddenly clarifies: **consumers are “connectors”** between pairs of provider modules. One could say that the Software System is greater than the sum of its modules, due to the interactions of the external consumer *connectors* with provider modules.

There are two slightly different situations with the case studies in this paper. The Command Design Pattern of the providers-only Modularity Matrix has the same number of three modules (Fig. 1) as the System Modularity Matrix with connectors (Fig. 10). The two connectors are external to the three modules and actually connect pairs of modules.

In the Asynchronous Server case study the providers-only Modularity Matrix (Fig. 11) is strictly diagonal and has 8 single-structor modules. In the System Modularity Matrix with connectors (Fig. 15) the providers-only modules were re-configured into three larger modules internally linked by connectors. In addition there are two external connectors. It is still true that connectors – both internal and external – link pairs of modules. The internal connectors link the original providers-only modules. The external connectors link the re-configured system modules (containing both providers and consumers, playing the internal connectors role).

We are led to the following conjecture as a summary of our currently empirical findings:

Conjecture: Software System Modularity Connectors

The *Minimal Number of System Module Connectors* is equal to the number of System Provider Modules minus one, i.e. each System module is connected to at least one other System module by a consumer Connector.

B. System Benefits for Software Design

First of all, it has been clear, before this work, that providers-only Modularity Matrix, the corresponding bipartite graph and its Laplacian Matrix, were incomplete descriptions of a software system. The addition of the consumers certainly improves the ability to judge the overall system design quality.

Once consumers are interpreted as “connectors” of provider modules, there are clear expectations on *consumers'* quantity and matrix element locations for their software system role.

Moreover, there are two System Matrix correctness criteria:

- a- **Algebraic** – the providers and consumers joint modules, excluding the external connectors, obey linear independence and block-diagonality;

- b- **Semantic** – system modules are semantically sound (e.g. the Fig. 15 lower-right module clusters *read*, *write*, and *stream* belong to the same category of messaging functionals).

C. Future Work

The paper's results, in particular the Software System Modularity Connectors conjecture, deserve formal proofs and extensive verification for a variety of software systems. These will be presented in an expanded version of this paper.

Although Fiedler (see e.g. [1]) extended the Laplacian spectral properties validity to weighted graphs, we need to investigate the specific weights' influence on modules calculations done with the Laplacian matrix.

D. Main Contribution

The main contribution of this paper is the introduction of Consumers in the software System Overall Modularity Matrix, and in the corresponding Laplacian Matrix, in the role of connectors between provider modules.

We have thereby shown that the same Linear Software Models that have been applied to providers-only matrices, is a generic algebraic theory of software composition, applicable to the overall system, including consumers.

ACKNOWLEDGMENT

The authors are grateful to the anonymous reviewers for incisive comments that helped to improve this version of the paper.

REFERENCES

- [1] N.M.M. de Abreu, "Old and new results on algebraic connectivity of graphs", *Linear Algebra and its Applications*, 423, pp. 53-73, 2007. DOI: <https://doi.org/10.1016/j.laa.2006.08.017>.
- [2] C.Y. Baldwin and K.B. Clark, *Design Rules*, Vol. I. The Power of Modularity, MIT Press, MA, USA, 2000.
- [3] Boost libraries, asio c++11 examples, Christopher M. Kohlhoff. URL: https://www.boost.org/doc/libs/1_66_0/doc/html/boost_asio/example/cpp11/echo/async_tcp_echo_server.cpp
- [4] T.Y. Browning, "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions", *IEEE Trans. Eng. Management*, Vol. 48, pp. 292-306, 2001.
- [5] Y. Cai and K.J. Sullivan, "Modularity Analysis of Logical Design Models", in *Proc. 21st IEEE/ACM Int. Conf. Automated Software Eng. ASE'06*, pp. 91-102, Tokyo, Japan, 2006.
- [6] G. Clark, S. Gilmore, J. Hillston and N. Thomas, "Experiences with the PEPA performance modelling tools", *IEE Proceedings, Software*, vol. 146, no. 1, pp. 11-19, 1999. DOI: <https://doi.org/10.1049/ip-sen:19990149>
- [7] CsharpDesignPatterns, by Jason de Oliveira, 2017. URL: <https://csharpdesignpatterns.codeplex.com/SourceControl/latest#DesignPatterns/DesignPatterns/DesignPatterns.csproj>
- [8] I. Exman, "Linear Software Models", Extended Abstract, in I. Jacobson, M. Goedicke and P. Johnson (eds.), *GTSE 2012, SEMAT Workshop on General Theory of Software Engineering*, pp. 23-24, KTH Royal Institute of Technology, Stockholm, Sweden, 2012. Video: <http://www.youtube.com/watch?v=EJfzArH8-ls>
- [9] I. Exman, "Linear Software Models are Theoretical Standards of Modularity", in J. Cordeiro, S. Hammoudi, and M. van Sinderen (eds.): *ICSOFT 2012, Revised selected papers, CCIS*, Vol. 411, pp. 203-217, Springer-Verlag, Berlin, Germany, 2013. DOI: [10.1007/978-3-642-45404-2_14](https://doi.org/10.1007/978-3-642-45404-2_14)
- [10] I. Exman, "Linear Software Models: Standard Modularity Highlights Residual Coupling", *Int. Journal on Software Engineering and Knowledge Engineering*, vol. 24, pp. 183-210, March 2014. DOI: [10.1142/S0218194014500089](https://doi.org/10.1142/S0218194014500089)
- [11] I. Exman, "Linear Software Models: Decoupled Modules from Modularity Eigenvectors", *Int. Journal on Software Engineering and Knowledge Engineering*, vol. 25, pp. 1395-1426, October 2015. DOI: [10.1142/S0218194015500308](https://doi.org/10.1142/S0218194015500308)
- [12] I. Exman and D. Speicher, "Linear Software Models: Equivalence of the Modularity Matrix to its Modularity Lattice", in *Proc. 10th ICSOFT'2015 Int. Conference on Software Technology*, pp. 109-116, ScitePress, Portugal, 2015. DOI: [10.5220/0005557701090116](https://doi.org/10.5220/0005557701090116)
- [13] I. Exman and R. Sakhnini, "Linear Software Models: Modularity Analysis by the Laplacian Matrix", in *Proc. 11th ICSOFT'2016 Int. Conference on Software Technology*, Volume 2, pp. 100-108, ScitePress, Portugal, 2016. DOI: [10.5220/0005985601000108](https://doi.org/10.5220/0005985601000108)
- [14] I. Exman and R. Sakhnini, "Linear Software Models: Bipartite Isomorphism between Laplacian Eigenvectors and Modularity Matrix Eigenvectors", *Int. Journal of Software Engineering and Knowledge Engineering*, Vol. 28, No 7, pp. 897-935, 2018. DOI: <http://dx.doi.org/10.1142/S0218194018400107>
- [15] M. Fiedler, "Algebraic Connectivity of Graphs", *Czech. Math. J.*, Vol. 23, (2) 298-305, 1973.
- [16] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, MA, 1995.
- [17] F.R. Gantmacher, *The Theory of Matrices*, Volume Two, Chelsea Publishing Co., New York, NY, USA, 1959. Chapter XIII, page 53, Available in the Web (out of copyright): <https://archive.org/details/theoryofmatrices00gant>.
- [18] M. Shtern and V. Tzerpos, "Clustering Methodologies for Software Engineering", in *Advances in Software Engineering*, vol. 2012, Article ID 792024, 2012. DOI: [10.1155/2012/792024](https://doi.org/10.1155/2012/792024)
- [19] M. Siff and T. Reps, "Identifying modules via concept analysis", *IEEE Trans. Software Engineering*, Vol. 25, (6), pp. 749-768, 1999. DOI: [10.1109/32.824377](https://doi.org/10.1109/32.824377)
- [20] R. Todd and E.W. Weisstein, "Connected Component", *Wolfram MathWorld*.
- [21] R. Wille, "Restructuring lattice theory: an approach based on hierarchies of concepts". In: I. Rival (ed.): *Ordered Sets*, pp. 445-470, Reidel, Dordrecht, Holland, 1982.
- [22] S.S. Yau and M.U. Caglayan, Distributed software system design representation using modified Petri Nets, *IEEE Trans. Software Engineering*, Vol. SE-9, pp 733-745, 1983.

An Effort Estimation Support Tool for Agile Software Development: An Empirical Evaluation

Emanuel Dantas^{§ ¶}, Alexandre Costa[¶], Marcus Vinicius[¶], Mirko Perkusich^{§ ¶},
Hyggo Almeida[¶], Angelo Perkusich[¶]

[§] Federal Institute of Paraíba

Monteiro, Paraíba, Brazil, 58500-000

{emanueldantas, mirkoperkusich}@ifpb.edu.br

[¶] Intelligent Software Engineering (ISE) Group, Federal University of Campina Grande
Campina Grande, Paraíba, Brazil, 58429-140

{marcusbarbosa, alexandrecoستا, almeida, perkusich}@embedded.ufcg.edu.br

[¶] CAPES Foundation, Ministry of Education of Brazil, Brasilia - DF, Zip Code 70.040-020

Abstract—Accurate effort estimation is an important part of the software process. In Agile Software Development, the techniques for predicting effort are mostly based on expert judgment, but there are approaches based on Machine Learning. The theme continues to be challenging and a subject of further studies given the difficulty of finding accurate solutions to the problem. This paper proposes and evaluates a tool based on the decision tree method for effort estimation in agile projects. We evaluated our tool given its accuracy and ease of use collecting data from four projects. To evaluate the accuracy, we compared the values of Magnitude of Relative Error from the teams' estimations with the values provided by the tool. To evaluate the ease of use, we used the Technology Acceptance Mode. The initial results show that the tool can be reliably used with minimal training. In terms of accuracy, the tool achieved lower error compared to the estimates provided by the teams (mean: 19.05% vs 33.32%), and the evaluation means in TAM were higher than 4.0 in ten of the eleven variables analyzed on a Likert scale. From this work, we conclude that estimation by decision tree is a viable technique that, at the very least, can be used by project managers to complement current estimation techniques.

Keywords—Agile Software Development; Effort Estimation; Machine Learning; Decision Tree.

I. INTRODUCTION

Since 2001, there has been an increasing interest in agile methodologies. Nowadays, these methods are widely used in software industry projects, especially, because of its volatility and flexibility. Agile software development (ASD) focuses on the needs of the rapidly changing environment by embracing the proposal of iterative and incremental development [7]. According to a recent study [2], Scrum stands out as an agile development process with 56% of preference. Scrum is a framework for developing and sustaining complex products [1]. One of its main events is the Sprint Planning Meeting in which occurs the task breakdown. During this process, the Scrum team defines tasks to be developed and estimates the effort to complete them.

In ASD context, effort estimation consists of predicting the effort needed to fulfill a given task [24], which is an important part of the development process, since it is one of the many steps that can lead to successful project completion. Due to

the dynamic nature of ASD [8] and limited documentation [36], effort estimation is considered a complex and critical task [4]. Traditionally, it depends heavily on the expert experiences [36] and the estimated values usually are far away from the real ones, resulting in low accuracy. On the other hand, accurate estimations can improve the development planning by enabling optimal assignments of both stories and developers [22]. Additionally, high accurate values can be used in different prediction models to improve the outputs. For instance, to increase project velocity [14], to optimize developer effort across different projects in the organization, and different projects inside the organization can be centrally coordinated to increase efficiency.

Over the past years, there has been a growing interest in effort estimation researches [21], [37], [32]. In previous work [9], we identified different approaches proposed to estimate effort using historical data and expert opinions based techniques. A significant amount of these used Artificial Intelligence or Machine Learning techniques to support effort estimation in ASD, which contributed to achieve higher accuracy. Despite the contributions of recent studies, low accurate prediction is still considered a gap in effort estimation.

In this study, we focus on the empirical evaluation of an effort estimation support tool for agile software development. The proposed tool is based on historical data and uses a Decision Tree to estimate effort during software development. In the evaluation, we aim to measure (i) the accuracy of the proposed tool and (ii) its usefulness and ease of use. To do so, we used Magnitude of Relative Error (MRE) to compare the estimations made by the project team members and those ones provided by the tool with the real effort (in hours). Furthermore, through a questionnaire based on the Technology Acceptance Model (TAM) [10], professionals with experience in software development used the tool and reported their perceived usefulness and perceived ease of use. As a result, the proposed tool achieved lower MRE values compared to ones provided by the teams (mean: 19.05% vs 33.32%). As also, most of the professionals who participated in the study found the approach to be useful and easy to use to support the estimation process (evaluation mean was higher than 4.0 in ten of the eleven variables analyzed on a Likert scale [16]).

The remainder of this paper is organized as follows: Section

II presents the background information. Section III details the proposed tool. Section IV presents the empirical evaluation, followed by the results and discussion in Section V. Finally, threats to validity and conclusions are described in Sections VI and VII, respectively.

II. BACKGROUND AND RELATED WORK

This section presents the background and related work on this paper. A considerable number of studies has been published on this subject. A Systematic Literature Review (SLR) [39] performed in 2014 aggregated and described the state of the art related to estimation techniques, effort predictors, accuracy measures and agile methods used. In 2018, Dantas et al. [9] updated this review and observed a strong indication of solutions based on Artificial Intelligence (AI) and Machine Learning (ML) methods for effort estimation in ASD. The purpose of the identified studies is to support human judgment during effort estimation. The main techniques observed were: Bayesian Network and Decision Tree.

In general, the studies use different factors in their approaches to estimate effort more accurately, either to support traditional estimation methods such as Planning Poker or in ML and AI solutions. These factors are known as Cost Drivers and represent personal or project factors that influence the estimated values [9]. For instance, to support Planning Poker, Lenarduzzi [19] considered technical ability, competence level, and managerial skill. While Grapenthin et al. [13] used factors related to the project: complexity and flexibility.

Artificial Intelligence is a branch of computer science formed by techniques that support activities of optimization or knowledge discovery [45]. A popular AI technique is the Bayesian Network which is used to estimate values considering the uncertainty of the variables (cost drivers) and can be constructed using historical data and experts. In the context of effort estimation, Mendes [25] proposed a Bayesian Network for Web effort estimation using knowledge from a domain expert. Karna and Gotovac [17] presented another model, including the relevant cost drivers with turnover, priority, and severity. Zahraoui et al. [46] adjusted story points using: priority, size, and complexity. The complexity and importance of a user story were considered by Lopez et al. [20], while Dragicevic et al. [11] considered the skills of the developers and requirements complexity. These approaches use historical data and expert knowledge to estimate effort with better accuracy.

Machine Learning is a current application of AI based on the idea that a system can learn from data rather than through explicit programming. Thus, the ML methods aim to improve the performance at certain tasks learning from the observed data. Many studies use ML to estimate effort in software projects. For instance, in Satapathy and Rath [32] Decision Tree, Stochastic Gradient Boosting and Random Forest were compared to assess effort estimation given a dataset composed by 21 projects. Porru et al. [29] used Support Vector Machine, K-Nearest Neighbors and Decision Tree for the same purpose given data from eight open source projects. Several Neural Networks were used in Panda et al. [27]. The purpose of these works is to support human judgment during effort estimation.

In this study, we use a Decision Tree which is a decision support method indicated for establishing classification or regression based on multiple covariates for developing prediction algorithms for a target variable [35]. This method constructs

prediction models by recursively partitioning the data and fitting a simple prediction model within each partition [6], till a halting paradigm is fulfilled [32]. For this purpose, it uses a sequential process to identify the predictor variables that best differentiate groups along with the outcome variable of interest.

Effort estimation is still attractive to researchers since a reliable estimation process is crucial for correct project planning and a good management of the resources [33]. Several other studies on the subject can be found in the literature, mainly focus on designing a good estimator [18], [3], new cost metrics [26] or other aspects [15].

III. PROPOSED TOOL

A. Overview

The effort estimation support tool presented in this paper uses historical data to construct a decision tree predictive model. The purpose of the tool is to support teams during task effort estimations. Therefore, it assists in the development process during the planning phase. The tool consists of a web application with an interface implemented in Angular that can be accessed from computers or smartphones. The key parts of the tool are discussed below.

B. Dataset

Agile projects often manage user requirements with models called User Stories (US). These artifacts are used to describe features that deliver value to the customer. USs are written in natural language, which makes it difficult to retrieve information [30]. Therefore, to build a dataset we have defined a model to represent the main features of software projects.

The model was built by specialists and is organized in a hierarchical structure of 03 levels to represent each feature. The first level is called *Module*, the second is *Operation*, and finally for each operation we have different types of *Tasks* in the third level. The whole model is composed by 03 *Modules*, 18 *Operations* and 131 *Tasks* types. Table I shows an example of operations defined to the "Authentication" module.

Table I: Examples of previously defined operations for Authentication module

Module	Operation
Authentication	Perform login with username and password
	Perform login with OAuth
	Password recovery
	First login
	Validate user permissions
	Update profile
	Create account
	Remove account

With the defined model, we analyzed the backlog of different organizations to create our dataset. All companies work with agile processes based on Scrum and develop projects on the web platform. We have analyzed 26 backlogs, containing a total of 530 USs and 1879 tasks. At the end, we were able to map 52% of tasks according to the model, which results in 977 items that form our dataset. For each record, in addition to the information representing the feature (*Module*, *Operation*, *Task*), we extract information from the following cost drivers: effort, human resource and technology. The information was taken from the teams during the planning meetings.

C. Prediction Algorithm

The proposed tool adopts the M5P algorithm for building the Decision Tree [43]. This algorithm represents an appropriate choice because it implements as much decision trees as linear regression for predicting a continuous variable. To train the algorithm, we used a 10-fold cross validation [5] and choose Effort (in hours) as the dependent variable. In Figure 1 we can see the relations of the cost drivers used to construct the predictive model. Category corresponds to the feature according to the model, human resource is the professional responsible. Finally, the predominant technology in the feature is also used to forecast the varying effort. Several Cost Drivers were analyzed using the Weka software, but these were chosen for achieving a result with the least error.

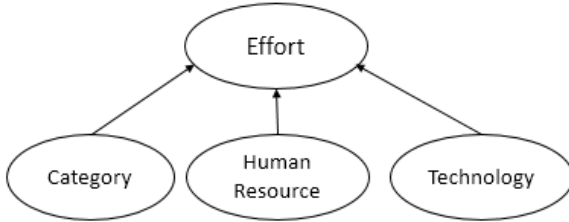


Figure 1: Cost Drivers Relation.

D. Design

The architecture of the proposed tool consists of a processing layer which executes the Decision Tree algorithm and a serving layer which exposes a REST API. The database has been implemented in MySQL and contains tables to store the historical data of the projects. To perform the processing, we used the class M5P¹ in Weka, version 3.8.0. Its interface was build using Angular framework, version 6.1.2. The tool is available² and can be accessed in any web browser.

The tool was developed for supporting agile teams during Sprint Planning. As shown in Figure 2, the tool has a view that shows all the tasks types and the structure is represented in three hierarchical levels (see the area of the screen indicated by 1). These tasks are in agreement with the model created in the construction of our dataset. Once selected the task type, the tool processes the decision tree algorithm and displays an effort estimation (see area of the screen indicated by 2). Finally, clicking on “detail”, the user can have access to the historical values of this task.

IV. EMPIRICAL EVALUATION

This section presents the case study performed to evaluate the proposed tool, which is detailed in the following sub-sections.

A. Case Study Planning

The evaluation is oriented towards design science in software engineering [42]. The improvement problem is the low accuracy from effort estimation tasks. The artifact used is a

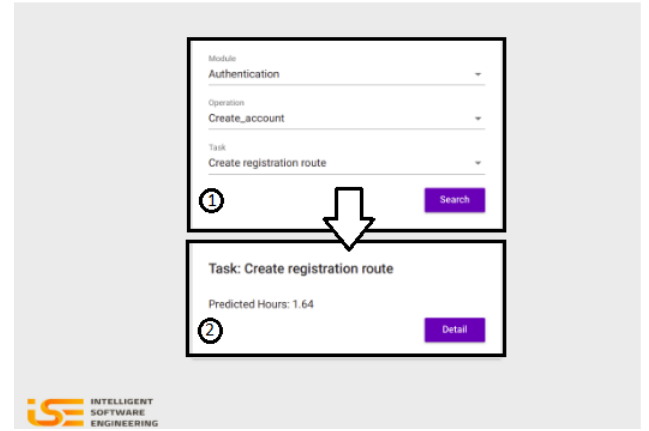


Figure 2: Screenshot of tool.

historical data based tool to support decision making during planning meetings. Therefore, we seek to answer the following research questions:

- RQ1: *Category*, *Human resource* and *Technology* can be used to construct a decision tree predictive model capable to reduce effort estimation errors during planning meetings?
- RQ2: The perceived usefulness and ease of use have a positive impact on the tool adoption?

To investigate the research questions, we conducted an exploratory case study [28] in a Brazilian software development company. Four agile software projects, based on Scrum, were available to participate in the study. Each project was composed of six developers and one Scrum Master who also performs the project manager role, each developer works exclusively on one project. According to the company practice, the Scrum Master does not participate in the effort estimation process, only the developers. All the projects belong to web development platform and were implemented with similar technologies.

The study lasted six sprints with 15 days each, resulting in three months of evaluation. Before the beginning, the participants were submitted to 15 minutes of training to get familiar with the proposed tool. At this moment, they received instructions and examples of use to learn how to utilize the tool.

Most participants were graduate developers (84.61%), who worked full time at the company and had more than three years of experience in software development. The remaining participants (15.49%) were undergraduate students from a computer science course, who worked part-time and had at least a year of experience in software development.

B. Case Study Execution

During the case study, the tasks were created by the teams and recorded in a spreadsheet. The authors of this paper were responsible for analyzing each task and for mapping them to the corresponded category of the model presented in Section III-B. Thereafter, the project teams estimated the effort of the tasks. At this moment, they could estimate the values by their own or use the tool to assist in the process. At the end of the study, 76% of the tasks were mapped, which corresponds

¹<http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/M5P.html>

²<https://mot-client-web.herokuapp.com/>

to 121 items. For each task, we recorded three values: (i) the team estimation, (ii) the tool estimation and (iii) the real effort.

To answer the RQ1, we choose the MRE (Equation 1) which measures the difference between real and estimated effort relative to the real one. Therefore, we compared the team MRE values with the tool MRE values, resulting in two value lists. Then, we use a confidence interval (with 95% confidence level) to do a statistical analysis. This procedure is recommended to characterize the uncertainty associated with a certain parameter estimated [41].

$$MRE = \frac{Real\ effort - Estimated\ effort}{Real\ Effort} \quad (1)$$

To answer the RQ2 we applied a questionnaire based on the indicators presented in the TAM [10]. The questionnaire aims at assessing users' beliefs about the usefulness and ease of use of a technology that is expected to support their work. It has been used extensively to explain and predict users' acceptance of information technology [34]. According to TAM, two variables impact adoption: perceived usefulness and perceived ease of use. Perceived usefulness refers to the degree to which an individual believes that using a particular technology would enhance his or her job performance. Perceived ease of use refers to the degree to which an individual believes that using a particular technology would be free of physical and mental effort [40]. Table II shows the applied questionnaire. The questions are divided by type: Perceived usefulness (PU), Perceived ease of use (PEoU), External variables (EV) and Attitude (AT). The type of response follows a Likert scale [16] with five possible answers ranging from strongly disagree (mapped to number 1) to strongly agree (mapped to number 5). The participants were asked to respond to each statement in terms of their own degree of agreement or disagreement [23]. The questionnaire was applied at the end of the six sprints and all the participants submitted their answers.

V. RESULTS AND DISCUSSION

This section outlines the results with respect to the research questions, which are answered in different sub-sections.

A. RQ1: Category, Human resource and Technology can be used to construct a decision tree predictive model capable to reduce effort estimation errors during planning meetings?

After collecting data from the case study, we were able to calculate MRE values from 121 tasks, resulting in two types of errors: team MRE and tool MRE. As mention before, we used the confidence interval to analyze the data. In several regression applications, these intervals are computed by supposing the errors follow a normal (Gaussian) distribution or another more general distribution with a number of parameters [31].

In Table III, are shown the mean, standard deviation and confidence intervals (with 95% confidence level) for the MRE values. We can observe that the tool has achieved lower MRE mean values when compared to each project team values. The overall result shows a 19.05% MRE mean value for the tool and 33.32% for the teams, i.e., the proposed tool can provide more accurate estimations. Also, the lower overall standard deviation obtained by the tool indicates that the estimated values tend to be close to the mean, generating more homogeneous

predictions. The overall confidence intervals (14.87% - 23.62% and 25.55% - 41.42%) do not intersperse, thus the two group values can be considered statistically different.

Regarded to the RQ1, we can conclude that the selected cost drivers can be used to construct a decision tree predictive model capable to reduce effort estimation errors during the planning meeting. Therefore, the proposed tool can provide accurate estimations when compared to the ones given by the project teams.

B. RQ2: The perceived usefulness and ease of use have a positive impact on the tool adoption?

In order to carry on a qualitative evaluation, we used the TAM. Table II shows the assigned variables for this study.

Table IV describes the values for median, mean, standard deviation (SD) for questions of the perceived usefulness. All average values were higher than 4.0, indicating that participants generally had positive attitudes toward the tool.

Table IV: Perceived usefulness

Variable	Definition	Mean	Median	SD
V1	Using the tool is useful for estimating task effort.	4.32	4	0.712
V2	Using the tool allows quick access to a historical basis	4.82	5	0.743
V3	The tool is accessed at all planning meetings	4.12	4	1.04

In Table V, all mean values related to the ease of use are above the midpoint and the standard deviations are within the range from 0.716 to 0.926 indicating a narrow spread around the mean.

Table V: Perceived ease of use

Variable	Definition	Mean	Median	SD
V4	Learning to use the tool was easy for me	4.74	5	0.786
V5	I often get confused in researching and understanding information in the tool	3.94	4	0.926
V6	Access to the tool is simple	4.78	5	0.716

The variable V9 in Table VI has the lowest mean value. This occur because the tool was recently adopted, thus the motivation among co-workers was expected to be low. Also, the standard deviation of V9 was greater than one, representing a high dispersion of the information.

Table VI: External variables

Variable	Definition	Mean	Median	SD
V7	The navigation features (menus, icons, links and buttons) are all clear and easy to find	4.34	4	0.696
V8	The tool has a nice interface	4.74	5	0.556
V9	My co-workers encourage me to use the tool	3.42	3	1.213

The results of Table VII show that the participants perceived the attitude of the information systems, since, for all variables, the average has resulted in more than 4.0, showing agreement on all the statements.

Table VII: Attitude

Variable	Definition	Mean	Median	SD
V10	I believe it's best to use the tool instead of traditional planning.	4.46	4	0.879
V11	My intention is to use the tool to better plan my project tasks	4.24	4	0.786

Table II: Questionnaire Statements on: Perceived usefulness, Perceived ease of use, External variables and Attitude

Type	Definition	Variables
Perceived usefulness (PU)	The level at which a person believes that using the tool improves the performance of their tasks.	V1: Using the tool is useful for estimating task effort. V2: Using the tool allows quick access to a historical basis V3: The tool is accessed at all planning meetings
Perceived ease of use (PEoU)	Level at which the person presents their perception of the tool in terms of ease of learning and operation.	V4: Learning to use the tool was easy for me V5: I often get confused in researching and understanding information in the tool V6: Access to the tool is simple
External variables (EV)	External variables provide a better understanding of what influences perceived utility and ease of use.	V7: The navigation features (menus, icons, links and buttons) are all clear and easy to find V8: The tool has a nice interface V9: My co-workers encourage me to use the tool
Attitude (AT)	Intention of the individual to use the tool	V10: I believe it's best to use the tool instead of traditional planning . V11: My intention is to use the tool to better plan my project tasks

Table III: Results of the MRE calculation

	Team MRE				Tool MRE			
	Mean (%)	Standard Deviation (%)	Lower Confidence Interval (%)	Upper Confidence Interval (%)	Mean (%)	Standard Deviation (%)	Lower Confidence Interval (%)	Upper Confidence Interval (%)
Project A	43.27	55.12	38.43	42.12	25.67	15.23	23.16	32.19
Project B	26.56	38.77	22.48	31.02	14.78	10.12	13.21	19.43
Project C	31.45	42.24	26.38	33.10	18.21	27.34	16.33	22.91
Project D	32.05	58.12	30.21	36.39	17.54	26.21	15.88	20.32
Overall	33.32	51.33	25.55	41.42	19.05	22.92	14.87	23.62

When using Likert-type scales it is imperative to calculate and report Cronbach's alpha coefficient for internal consistency reliability [12]. The Cronbach's alpha is an index of inter-item homogeneity that describes how related a set of items is as a group [38]. In table VIII, we can see the coefficient for each variable. All multi-item constructs should meet the guidelines for a Cronbach's alpha of greater than 0.70. Table VIII demonstrates that the coefficient values, for all constructs in the measurement model, exceeded the recommended threshold. Therefore, based on the presented results related to the RQ2, we can conclude that the tool is easy to use and has utility for the teams.

Table VIII: Descriptive statistics

Construct	Variables	Mean	Cronbach's
Perceived usefulness	V1	4.32	0.8829
Perceived usefulness	V2	4.82	0.9084
Perceived usefulness	V3	4.12	0.8779
Perceived ease of use	V4	4.74	0.8745
Perceived ease of use	V5	3.94	0.8238
Perceived ease of use	V6	4.78	0.8779
External variables	V7	4.34	0.8989
External variables	V8	4.74	0.8779
External variables	V9	3.42	0.8229
Attitude	V10	4.46	0.9208
Attitude	V11	4.24	0.8779

VI. THREATS TO VALIDITY

This section presents the main threats to validity identified in the study, according to Wohlin et al. [44]. The analysis of the threats aims to examine the relationship between the conclusions reached and the reality.

A threat to external validity identified is related to the platform of the selected projects. All of them were web development projects with similar technologies, thus our findings can not be generalized. To address this threat, we intend to conduct a new case study with projects from mobile and desktop platforms. Another identified threat refers to internal validity. The teams that participated in the case study were composed by graduate developers and undergraduate students, with different levels of experience, which can impact the error measurement. Therefore, the results from this study must be considered as indicators and further evaluation with different contexts must be conducted.

VII. CONCLUSION AND FUTURE WORK

This paper proposed and evaluated an effort estimation support tool for agile software development. The tool uses historical data to construct a decision tree predictive model to provide effort estimations during the planning meeting. To evaluate the tool, we performed a case study in a Brazilian software development company, in which four agile projects (based on Scrum) and 24 professional participated. The case study lasted three months and thereafter we performed a quantitative and qualitative analysis.

The achieved results indicate that the tool can support the planning meeting, reducing the errors related to the task estimation process. Also, we concluded that the majority of the professionals who participated in the study found the tool useful and easy to use for supporting the effort estimation, and they would regularly use the tool for future plannings in their job.

As a limitation, we realize that the low numbers of teams and the types of projects are prohibitive to generalize our findings. In the future, we plan to conduct another case study with more projects and different contexts. Additionally, we would like to evaluate the usage of the tool with more experienced teams.

ACKNOWLEDGMENT

The authors acknowledge the financial support given by CAPES/Brazil.

REFERÊNCIAS

- [1] Sutherland j, schwaber k. the scrum guide. <https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf>. accessed in 23/02/2019.
- [2] Versionone. 9th annual state of agile development survey results. <https://www.versionone.com/pdf/state-of-agile-development-survey-ninth.pdf>. accessed in 23/02/2019.
- [3] M. A. Ahmed, I. Ahmad, and J. S. Alghamdi. Probabilistic size proxy for software effort prediction: A framework. *Information and Software Technology*, 55(2):241–251, 2013.
- [4] M. S. S. Basha and D. Ponnuram. Analysis of empirical software effort estimation models. *CoRR*, abs/1004.1239, 2010.

- [5] N. Bhargava, S. Dayma, A. Kumar, and P. Singh. An approach for classification using simple cart algorithm in weka. In *Intelligent Systems and Control (ISCO), 2017 11th International Conference on*, pages 212–216. IEEE, 2017.
- [6] L. Breiman. *Classification and regression trees*. Routledge, 2017.
- [7] M. Brhel, H. Meth, A. Maedche, and K. Werder. Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology*, 61:163–181, 2015.
- [8] A. Cockburn. *Agile Software Development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2007.
- [9] E. Dantas, M. Perkusich, E. Dilorenzo, D. Santos, H. Almeida, and A. Perkusich. Effort estimation in agile software development: an updated review. In *Proceedings of the 30th International Conference on Software Engineering and Knowledge Engineering*, 2018.
- [10] F. D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, pages 319–340, 1989.
- [11] S. Dragicevic, S. Celar, and M. Turic. Bayesian network model for task effort estimation in agile software development. *Journal of Systems and Software*, 127:109–119, 2017.
- [12] J. A. Gliem and R. R. Gliem. Calculating, interpreting, and reporting cronbachs alpha reliability coefficient for likert-type scales. Midwest Research-to-Practice Conference in Adult, Continuing, and Community Education, 2003.
- [13] S. Grapenthin, M. Book, T. Richter, and V. Gruhn. Supporting feature estimation with risk and effort annotations. In *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*, pages 17–24. IEEE, 2016.
- [14] P. Hearty, N. Fenton, D. Marquez, and M. Neil. Predicting project velocity in xp using a learning dynamic bayesian network model. *IEEE Transactions on Software Engineering*, 35(1):124–137, 2009.
- [15] M. Jørgensen. The influence of selection bias on effort overruns in software development projects. *Information and Software Technology*, 55(9):1640–1650, 2013.
- [16] A. Joshi, S. Kale, S. Chandel, and D. Pal. Likert scale: Explored and explained. *British Journal of Applied Science & Technology*, 7(4):396, 2015.
- [17] H. Karna and S. Gotovac. Estimating software development effort using bayesian networks. In *Software, Telecommunications and Computer Networks (SoftCOM), 2015 23rd International Conference on*, pages 229–233. IEEE, 2015.
- [18] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Transactions on Software Engineering*, 38(2):425–438, 2012.
- [19] V. Lenarduzzi. Could social factors influence the effort software estimation? In *Proceedings of the 7th International Workshop on Social Software Engineering*, pages 21–24. ACM, 2015.
- [20] J. López-Martínez, R. Juárez-Ramírez, A. Ramírez-Noriega, G. Licea, and R. Navarro-Almanza. Estimating user stories complexity and importance in scrum with bayesian networks. In *World Conference on Information Systems and Technologies*, pages 205–214. Springer, 2017.
- [21] J. López-Martínez, A. Ramírez-Noriega, R. Juárez-Ramírez, G. Licea, and S. Jiménez. User stories complexity estimation using bayesian networks for inexperienced developers. *Cluster Computing*, pages 1–14, 2017.
- [22] O. Malgonde and K. Chari. An ensemble-based model for predicting agile software development effort. *Empirical Software Engineering*, pages 1–39, 2018.
- [23] J. McIver and E. G. Carmines. *Unidimensional scaling*, volume 24. Sage, 1981.
- [24] E. Mendes. *Cost Estimation Techniques for Web Projects*. IGI Global, Hershey, PA, USA, 2007.
- [25] E. Mendes. Knowledge representation using bayesian networks a case study in web effort estimation. In *Information and Communication Technologies (WICT), 2011 World Congress on*, pages 612–617. IEEE, 2011.
- [26] T. Menzies, D. Port, Z. Chen, J. Hihn, and S. Stukes. Validation methods for calibrating software effort models. In *Proceedings of the 27th international conference on Software engineering*, pages 587–595. ACM, 2005.
- [27] A. Panda, S. M. Satapathy, and S. K. Rath. Empirical validation of neural network models for agile software effort estimation based on story points. *Procedia Computer Science*, 57:772–781, 2015.
- [28] M. Q. Patton. *Qualitative research evaluation methods*. SAGE Publications, Inc; 3rd edition, 2003.
- [29] S. Porru, A. Murgia, S. Demeyer, M. Marchesi, and R. Tonelli. Estimating story points from issue reports. In *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, page 2. ACM, 2016.
- [30] M. Robeer, G. Lucassen, J. M. E. van der Werf, F. Dalpiaz, and S. Brinkkemper. Automated extraction of conceptual models from user stories via nlp. In *Requirements engineering conference (RE), 2016 IEEE 24th international*, pages 196–205. IEEE, 2016.
- [31] F. Sarro, A. Petrozziello, and M. Harman. Multi-objective software effort estimation. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 619–630. IEEE, 2016.
- [32] S. M. Satapathy and S. K. Rath. Empirical assessment of machine learning models for effort estimation of web-based applications. In *Proceedings of the 10th Innovations in Software Engineering Conference*, pages 74–84. ACM, 2017.
- [33] E. Scott and D. Pfahl. Using developers’ features to estimate story points. In *Proceedings of the 2018 International Conference on Software and System Process, ICSSP ’18*, pages 106–110, New York, NY, USA, 2018. ACM.
- [34] J. Silva, D. Ramos, and M. S. Soares. On criteria to choose a content management system: A technology acceptance model approach. In *SEKE*, 2016.
- [35] Y.-Y. Song and L. Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015.
- [36] B. Tanveer, L. Guzmán, and U. M. Engel. Understanding and improving effort estimation in agile software development: An industrial case study. In *Proceedings of the International Conference on Software and Systems Process, ICSSP ’16*, pages 41–50, New York, NY, USA, 2016. ACM.
- [37] B. Tanveer, L. Guzmán, and U. M. Engel. Effort estimation in agile software development: Case study and improvement framework. *Journal of Software: Evolution and Process*, 29(11):e1862, 2017.
- [38] M. Tavakol and R. Dennick. Making sense of cronbach’s alpha. *International journal of medical education*, 2:53, 2011.
- [39] M. Usman, E. Mendes, F. Weidt, and R. Britto. Effort estimation in agile software development: A systematic literature review. In *Proceedings of the 10th International Conference on Predictive Models in Software Engineering, PROMISE ’14*, pages 82–91, New York, NY, USA, 2014. ACM.
- [40] L. G. Wallace and S. D. Sheetz. The adoption of software measures: A technology acceptance model (tam) perspective. *Information & Management*, 51(2):249–259, 2014.
- [41] L. Wasserman. All of statistics: a concise course in statistical inference. 2013.
- [42] R. J. Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.
- [43] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.
- [44] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.
- [45] S. Zahraee, M. K. Assadi, and R. Saidur. Application of artificial intelligence methods for hybrid energy system optimization. *Renewable and Sustainable Energy Reviews*, 66:617–630, 2016.
- [46] H. Zahraoui and M. A. J. Idrissi. Adjusting story points calculation in scrum effort & time estimation. In *Intelligent Systems: Theories and Applications (SITA), 2015 10th International Conference on*, pages 1–8. IEEE, 2015.

Towards an artifact to support agile teams in software analytics activities

Joelma Choma*, Eduardo M. Guerra*, Tiago Silva da Silva[†], Luciana A. M. Zaina[‡], Filipe Figueiredo Correia^{§¶}

*National Institute for Space Research, São José dos Campos, Brazil

Email: jh.choma@hotmail.com, guerraem@gmail.com

[†]Federal University of São Paulo, São José dos Campos, Brazil

Email: silvadasilva@unifesp.br

[‡]Federal University of São Carlos, Sorocaba, Brazil

Email: lzaina@ufscar.br

[§]Faculty of Engineering, University of Porto, Porto, Portugal

[¶]INESC TEC, FEUP Campus, Porto, Portugal

Email: filipe.correia@fe.up.pt

Abstract—Software analytics supports data-driven decision making, which allows software practitioners to leverage valuable insights from software data to improve their processes and many quality aspects of the software. In this paper, we present an artifact designed from a set of patterns to support agile teams to plan and manage software analysis activities, named Software Analytics Canvas. Further, we report the study undertaken to evaluate the ease of use and the utility of our canvas from the practitioners’ viewpoint, and a participatory design session carried out to collect information about possible artifact improvements. In general, subjects found the artifact useful, but some of them reported difficulties in learning and understanding how to use it. In the participatory design, they pointed out improvement points and a new layout for the canvas components. The results of both studies helped us refine the proposed artifact, improving both the terms used in each element and the layout of the blocks to make more sense for its users.

Index Terms—software analytics, agile teams, decision making, software data

I. INTRODUCTION

An increasing number of companies around the world have used data analytics to make decisions about their businesses. Analytics refers to the use of analysis, data, and systematic reasoning to inform the decision making process [1].

Nowadays, researchers and practitioners already use analytics applied to software data for better decision making concerning many aspects of software quality and its development process [2]. Zhang et al. [3] coined the term “Software Analytics” (SA) to label research in this area. Since then, SA has been widely adopted by large companies. However, it has not yet reached its full potential for broad industrial adoption. For small companies, for example, SA is an open question and rarely addressed [4]. Furthermore, to the best of our knowledge, there is no consolidated approach on how to introduce software analytics concepts and practices into an agile development context.

By emphasizing short feedback cycles, agile teams regard changes as an opportunity to improve the product at any time

in the development process, always focusing on delivering value [5]. In this sense, software analytics can support agile teams to make more appropriate changes based on actual data, rather than only on their personal experiences or intuitions. Additionally, the adoption of data-driven continuous improvement can help agile teams to save resources and decrease the cost of building and maintaining the software [6]. Process improvement is one of the main reasons for measurement in agile software development [7]. However, the measurement tends to be immediate and straightforward, since agile principles emphasize measuring progress in terms of working software over measuring intermediate work products [5].

Currently, various tools provide structure on top which data-driven improvement processes can be implemented. However, due the perceived complexity and effort required to set up such tools and to establish a measurement program, agreed with the urgency of the product delivery, lack of time, and others reasons, many agile teams end up not systematically using metrics to track product and process performance [8]. There is a lack tools to ease the adoption of software analytics, but also a lack of approaches to effectively change the habits within agile teams towards data-driven decision making.

Seeking a practical method to introduce *software analytics* into agile teams, we have outlined an artifact to support software analytics activities in agile environments named Software Analytics Canvas (SA Canvas). It was designed based on a set of patterns that we have identified from experience reports of researchers and practitioners in the software analytics area [9], [10]. In this paper, we present the canvas and report two studies designed to (i) evaluate the usefulness and ease-of-use perceived by the users, and (ii) refine the design of the proposed artifact. The goal of these studies is to answer the following research questions (RQs):

- RQ1: What are the users’ perceptions about the usefulness and ease-of-use of the SA Canvas?
- RQ2: What characteristics can be improved in the design of the SA Canvas?

To answer RQ1, we recruited six software practitioners for planning and managing activities from a software analytics project using the canvas. As input to the project, we provide information from a web-based system in the Space Weather area. After three iterations with the canvas, the participants were able to evaluate the usefulness and ease of use of the artifact. To answer RQ2, we call the same subjects for a participatory design session to provide ideas for possible artifact enhancements.

II. BACKGROUND

A. Software Analytics

Software analytics allows software practitioners (developers, testers, designers, and managers, to name a few) to leverage insightful information (accurate and in-depth) and actionable (with real practical value) for completing various tasks around software systems, software users, and software development processes [11]. The SA process comprehends monitoring, analysis, and understanding of software data to support the decision-making process throughout the different phases of the software lifecycle [3].

Within the SA context, we refer to the software data as any data generated from different sources such as code, bug reports and test executions recorded in software repositories (e.g., version control systems and issue-tracking systems), and information about data of usage typically stored in the log files [12]. Furthermore, SA has been used to address different type of concerns, such as issues related to the source code (e.g., code quality, bug proneness, number of defects, and amount of effort to fix bugs) [13]; development process (e.g., productivity and ROI) [4]; product business (e.g., usage of features, data quality, and user satisfaction) [14]; and software runtime properties (e.g., performance, number of transactions and error log) [15].

B. Software Analytics Patterns

Considering software analytics as an essential practice for leveraging value delivery in agile contexts, we have identified in previous studies a set of eight patterns focusing on how to incorporate SA activities into agile practices on a continuous basis to inform the decision-making process of software practitioners, including project managers, analysts and software developers from small, large, or multiple teams. Below, we present the eight SA patterns and a brief description of the proposed solutions in each of them. For a more detailed description of these patterns, see [9] and [10].

- 1) **What You Want to Know:** refers to defining the key issues that the development team wants to focus on, in order to guide their selection of the appropriate means for measurement, assessment and monitoring these issues throughout the project.
- 2) **Choose the Means:** refers to defining the data sources and most appropriate means, such as tools, techniques and other approaches for selecting and collecting data that will be useful for future decisions.

- 3) **Software Analytics Planning:** refers to adding tasks related to the *software analytics* on the to-do list to be prioritized with the regular project tasks according to the team's demand for information.
- 4) **Analytics in Small Steps:** it means that the tasks related to the *software analytics* can be distributed throughout the project, adding information to the team about the system at small portions by adjusting the granularity of the analytic activities.
- 5) **Reachable Improvement Goals:** refers to defining reachable improvement goals from the *software analytics* findings and break the activities down into smaller tasks to fit together with the other tasks.
- 6) **Learning from Experiments:** refers to create an alternative solution and perform an experiment collecting data that allow the comparison with the current solution.
- 7) **Define Quality Standards:** refers to define quality standards and establish minimal or maximum thresholds for any software aspect that the team intends to monitor.
- 8) **Suspend Measurement:** refers to suspend measurement of the issues with a low possibility of recurrence, or measurement of the issues that need to be continually monitored, but the team has defined that, for some reason (e.g., effort, cost or other project constraints), the monitoring of these issues cannot be implemented immediately.

III. RESEARCH APPROACH

In order to develop an artifact to support agile teams throughout the software analytics activities, our research approach follows the guidelines of Design Science Research (DSR), as proposed by Hevner et al. [16]. DSR is a problem-solving paradigm, where “knowledge and understanding of a problem domain and its solution are achieved in the building and application of the designed artifact”.

In this paper, we describe the design process of the SA Canvas artifact, which includes the building and evaluation cycles. A first version of the canvas emerged from patterns previously identified from experiences reports in the SA field (Section II-B). Aiming at evaluating the artifact and improving its characteristics, we have carried out a formative evaluation in two rounds (Section V).

For the first round, we selected six subjects (in pairs) to plan and manage software analysis activities using the SA Canvas. After the hands-on experience of handling the artifact, we gathered the users' perceptions of usefulness and ease of use of the proposed artifact, using the Technology Acceptance Model (TAM) [17]. According to Davis [17], *perceived usefulness* (PU) refers to “the degree to which a person believes that using a particular system would enhance his or her job performance”; and *perceived ease of use* (PEU) refers to “the degree to which a person believes that using a particular system would be free of effort”. We measured both variables within the TAM through a multiple-item questionnaire using a 6-point Likert scale – from “Completely Disagree” to “Completely Agree”.

For the second round, we organized a redesign session using principles of participatory design (PD) [18] to collect information for artifact improvement, involving the same participants of the first round. During the PD session, the participants were encouraged to draw sketches as an idea for the canvas redesign.

IV. SOFTWARE ANALYTICS CANVAS DESIGN

This section presents the SA Canvas template we have designed as an artifact to support the software analytics activities into agile development environments. Canvas artifacts are visual maps – structured and preformatted – used to support the collaborative teamwork in their communication processes. Canvas is considered a hands-on tool that fosters understanding, discussion, creativity, and analysis on a given matter [19]. Nowadays, there is a range of applications using canvas – e.g., development of new businesses, conception, and planning of projects, strategic alignment of projects, value proposition, among others.

The SA patterns presented in Section II-B were the basis for the creation of our canvas. As shown in Figure 1, it is composed of seven blocks with names based on the patterns. The seven names are Key Issues, Data Sources, Data Gathering, Analytics Implementation, Insights, Incremental Goals, and Quality Thresholds. Note that, we added in each block a guiding question to help beginner users.

SOFTWARE ANALYTICS CANVAS		PROJECT: PROJECT NAME		VERSION: 1.0	
KEY ISSUES What do we want to know about the software, process and/or usage patterns?	DATA SOURCES What are the data sources that can provide information on the issues raised?	DATA GATHERING How to collect and analyze software data related to this issue?	INSIGHTS What have we found out from the analysis?	QUALITY THRESHOLDS What parameters we can establish to evaluate the quality of our decisions?	
ANALYTICS IMPLEMENTATION How to implement software analytics tasks along with other tasks?		INCREMENTAL GOALS What incremental goals can we establish based on the insights from data analysis?			
To Do Done		To Do Done			

Fig. 1. Software Analytics Canvas [1st Version]

The explanation of each canvas block is presented below, including its description, the related patterns, and its corresponding guiding question.

- **Key Issues.** As a first step, the team can raise issues that need to be verified, analyzed and improved. As mentioned in Section , the issues can be, for example, related to the internal quality of the system (e.g., code quality), external quality of the system (e.g., performance, bug density, the effort required to fix defects), productivity (e.g., effort estimation), and/or usage patterns (e.g., usability, user satisfaction). This element is related to the pattern called *What You Want to Know*. The guiding question is: *What*

do we want to know about the software, process and/or usage patterns?

- **Data Sources.** After defining the key issues, the team identifies what kind of data is needed to know more about the issue raised, and from which sources the data should be extracted, for example, a dump of the database system on recent transactions, source code, behaviors' user, historical data about bugs incidence, etc. For specific issues, the team may recognize the need to collect data from multiple sources for cross-referencing. The two patterns related to this element is *Choose the Means* and *Learning from Experiments*. The guiding question is: *What are the data sources that can provide information on the issues raised?*
- **Data Gathering.** After identifying the data sources, the team should decide which metrics and tools will be used to gather the data. The team can, for example, enable the collection of specific code metrics in the development environment, export from SGBD data referring to a given period, verify the need to create a specific script to extract data from the software repository, etc. Furthermore, the team needs to decide which methods and tools will be used to analyze the data collected. The team can employ from simple statistical methods (e.g., descriptive and inferential statistics) to more sophisticated methods (e.g., data mining, natural language processing, machine learning, etc.), according to the type and amount of data. Also, the team also needs to decide on the tools to support their analysis. For example, the team can opt for *software analytics platforms* which can be configured according to the team's needs. This element also is related to both *Choose the Means* and *Learning from Experiments*. The guiding question is: *How to collect and analyze software data related to this issue?*
- **Insights.** The team analyzes the results obtained from the collected data and discusses possible solutions and insights to making-decision. For example, the team find that "tests have low coverage in module X", "customers prefer this approach" and so on. Then, Insights are raised from the search for solutions. Notice that, sometimes, the data analysis did not reveal significant information about the issue raised. So, the team will decide whether to continue the investigation by collecting new data or if the issue is disregarded, once no action is necessary. This element is a trigger for *Reachable Improvement Goals* pattern. The guiding question is: *What have we found out from the analysis?*
- **Quality Thresholds.** When implementing the improvements via informed decision-making, the team can evaluate the impact of the changes by collecting feedback from stakeholders. From collecting feedback, the team will have enough information to decide whether to consider the issue resolved, or whether the issue should be monitored for longer. Concerning unresolved issues, the team will decide whether they will be re-analyzed from new data, or discarded. Ideally, the team should establish

the quality thresholds values for any issue that the team decides to evaluate or to keep in monitoring. For example, in issues related to coverage testing, the response time cannot exceed 2 seconds, or the test coverage must be at least 80%. *Define Quality Standards* is the patterns related to this element. The question is: *What parameters we can establish to evaluate the quality of our decisions?*

- **Analytics Implementation.** The team should plan how to conduct analysis activities to seek meaningful information from collected data. These activities should be included on the to-do list and prioritized along with the other development tasks. In order to avoid overloading the team, such activities – that also includes the preparation of the analytical infrastructure – can be distributed throughout the project and executed by steps resulting in something deliverable. This block is divided into two regions, the first for the works to be done, and the other to control the work done. *Software Analytics Planning* and *Analytics in Small Steps* are the patterns related to this element. The guiding question is: *How to implement software analytics tasks along with other tasks?*
- **Incremental Goals.** From their insights, the team discusses and define reachable goals to put their solutions into practice, considering that these improvements can be made incrementally. Therefore, the most important in this step is to define where the team wants to reach and what goals they want to achieve. *Reachable Improvement Goals* is the related pattern to this element. However, if the current goals have been fulfilled, the pattern is *Suspend Measurement*. The main question is: *What incremental goals can we define based on the insights from data analysis?*

V. ARTIFACT EVALUATION

This section describes the two studies conducted to evaluate the SA Canvas built to aid agile teams during the planning and managing of software analytics activities.

A. Perceived Usefulness and Ease-of-use (RQ1)

We have conducted the first study in order to investigate the users' perceptions of the usefulness and ease-of-use of the SA Canvas (RQ1). This study was conducted in the Laboratory of Computation and Applied Mathematics at the National Institute for Space Research in Brazil, where we selected six subjects from the course of Agile Projects. Participants had at least three years of experience in software development, and only one of them had no experience with agile methods. We divided the participants into three pairs, seeking to balance them based on the experience of each participant. Table I shows the participants' background and their experience time in developing software (in years).

During the study, participants were invited to plan and manage a software analytics project based on a real case using SA Canvas. For this purpose, we prepared for each team a SA Canvas using a whiteboard and a document describing the case study which should be used as input for the planning

TABLE I
PARTICIPANTS CHARACTERIZATION

Pairs	Subjects	Background	Professional Experience
G1	P1	web development	6 to 10
	P2	software architecture	16 to 20
G2	P3	systems analysis	3 to 5
	P4	systems analysis	11 to 15
G3	P5	computer programming	3 to 5
	P6	systems analysis	16 to 20

and management of the software analytics project. As for the case study, we describe a real case of EMBRACE, one of the research centers in the area of space weather at INPE. In the document, we described the context of study related to a web system to make available some of the products developed by the researchers working in the EMBRACE. In addition to the information about the products developed, we included in the document some concerns related to the web portal that the researchers of the space weather had the intention to investigate and solve. Considering the possible demands described in the document for a software analytics project, we prepared a list of possible data sources to give participants some examples. Moreover, then, we created a tutorial printed on A4 with the illustration of the canvas, the description of its components, and the guiding questions (in Portuguese).

We conducted a pilot study with two researchers from the computer lab to evaluate their iteration with the SA Canvas on the whiteboard and the other materials – case study description, list of possible data sources, and canvas tutorial. From the pilot study, (i) we established that peers should have to raise at least two issues of software analytics; and that (ii) we should carry out a warm-up exercise together with all participants before they begin planning their projects. After that, we held four weekly meetings with participants, as showed in Figure 2.

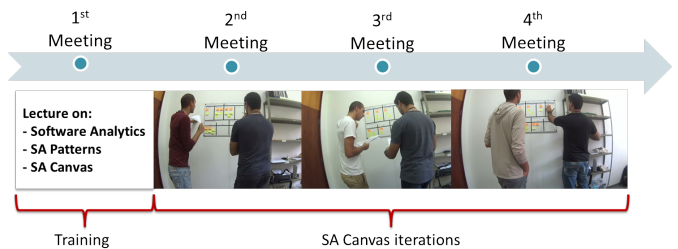


Fig. 2. Meetings over time.

1st Meeting. At the first meeting, we introduced concepts about software analytics, provided an overview of SA patterns, and presented SA Canvas using the practical example related to code coverage improvement. For more information, we asked the participants to read the two articles on software analytics patterns [9] [10] before the next meeting. After the meeting, participants read and signed an informed consent form to participate in this study.

2nd Meeting. At the second meeting, we invited the participants to a practical exercise to understand how they should use the canvas. One of the participants provided us with a real example of his own work. From this example, we helped them to plan software analytics activities on the board using sticky notes. This activity took approximately one hour.

After warming up, a pair of participants at a time should fill out four blocks of the Canvas (Key Issues, Data Sources, Data Gathering, and Analytics Implementation) based on information written in the document on the EMBRACE case study and the support material (list of possible data sources and the canvas tutorial). Additionally, we asked teams to use different color sticks for different key issues. Each pair took from 25 to 40 minutes to fill the four blocks with at least two key issues. The sessions were recorded for future analyzes, and one of the researchers observed the activities in silence, taking notes on the interaction of participants with the artifact.

After the first iteration, we prepared a report with (i) feedback about how they had used the canvas (including corrective actions), and (ii) fictional information on the evolution of the activities planned by them and executed by the developers. Also, we have introduced some insights mixed into the text to be extracted by them. Because each team raised different key issues, we then prepared three different documents for each of them.

3rd Meeting. The report's information was used as input for the second iteration when the team should adjust some fill-in mistakes pointed out by researchers, update the canvas with the tasks done, proceed in the planning their activities for the next iteration, and fill in the remaining canvas blocks from the insights. As in the first iteration, the sessions were recorded and observed by one of the researchers. The sessions took from 40 to 60 minutes. From the results of this iteration, we prepared a new report for each team with the feedback on the use of the canvas, other fictitious information about the evolution of the project, and new insights mixed into the text.

4th Meeting. As in the previous iteration, from the researchers' report, the teams should adjust some fill-in mistakes pointed out by researchers, update the canvas with the tasks done, proceed in the planning their activities for the next iteration, and fill in the remaining canvas blocks from the insights. The sessions took on average 40 minutes, and just like the previous ones, they were recorded and observed by one of the researchers. At the end of the iteration, we asked the participants individually to answer a questionnaire with questions of the TAM [17], based on their experiences during the software analytics project when they used SA Canvas to plan and manage its activities.

Findings. The questions answered by the participants can be found in the following link: <https://goo.gl/ZbEcC3>. Figure 3 shows the responses from the questionnaire, where to some degree all participants agree that the SA Canvas is useful. However, there was some disagreement concerning the ease of use. Some participants do not agree that it was easy to learn and understand what should be done at certain times (PEU1

and PEU2). The learning curve certainly will have an impact on the user ability to handle the artifact (PEU5). Furthermore, users tend to be more critical as to the usefulness of the artifact when its use is not effortless.

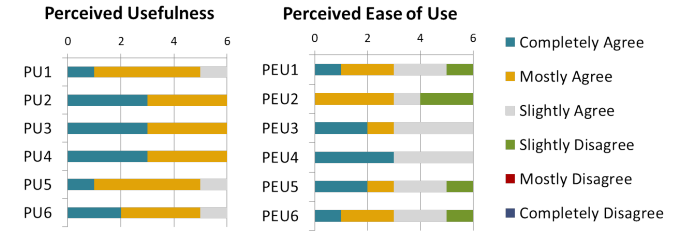


Fig. 3. Perceived usefulness and use of use.

B. Participatory Redesign (RQ2)

To identify what and how we could improve in the SA Canvas (RQ2), we conducted a participatory design session with the same participants from the previous study. The method used to conduct the redesign section was divided into three steps. In the first step, each participant received a document with the description of all the components of the canvas and a form to inform their particular suggestions to improve SA Canvas. After that, each participant should sketch a redesign proposal. In the second step, they consolidated their opinions with their peers by drawing a new sketch for canvas redesign. Lastly, in the third step, all participants discussed their redesign proposals and proposed a single sketch to the canvas. Two of the researchers participated in the session as observers making notes of relevant information. On average, each step lasted 30 minutes.

Findings. From the participants' suggestions, we can identify the need to better clarify at least three blocks:

- In "Insights", three participants suggested making it clear that insights should be described from the results of the analysis.
- The name of the "Quality Thresholds" block seems not to be suitable. Participants suggested making it clear that the values may be minimum or maximum.
- Understanding what should be considered in "Incremental Goals" was difficult for them. Some have suggested a more appropriate name since the main idea is to implement the improvements.

As a final result, the participants sketched the canvas with the following characteristics:

- At the top of the canvas, there are five blocks reserved for planning (inputs and outputs): "Key Issues", "Data Sources", "Data Gathering", "Incremental Goals", and "Quality Threshold".
- The "Quality Threshold" block has been subdivided into two parts to include a minimum acceptable value (minimum) and the goal to achieve (goal).
- At the bottom of the canvas, there is a block called "Analytics Tasks" block instead of "Analytics Implemen-

tation". This block has been subdivided into four parts to accommodate the tasks to do, in progress, done, and the impediments.

VI. DISCUSSION, CONCLUSION AND FUTURE WORK

Although feedback, continuous improvement, inspection, and adaptation are always present in the agile speech, there is little evidence on measurement in practice. Often this measurements and adaptation are adopted on an ad hoc basis. Software analytics is an approach that aims at data-driven continuous improvement, but it is not very widespread in practice.

To bridge this gap, we previously identified patterns on how software analysis could be introduced into the software development process continuously. By looking for a more efficient and practical way to apply the SA patterns into agile development context, we proposed a canvas addressed to the software analytics activities taking into account how teams communicate and collaborate. SA Canvas can be considered a goal-focused approach, similar to the well-known Goal Question Metric (GQM) approach proposed by Basili et al. [20]. The goal-focused approaches are good ways to ensure that measurement goals are articulated with the metrics being collected, and also, to avoid having useless measurements.

We argue that the SA Canvas is a suitable artifact to agile teams' informative workspaces where various techniques and tools for software visualization are commonly applied as information radiators – e.g., count of velocity automated tests, continuous integration status, incident reports, and so forth [21].

In our viewpoint, our canvas has two interesting characteristics. First, it can work as a hub in terms of information flow related to software analytics projects. Information hubs can be considered spaces where information flows meet and decisions are made. Second, the proposed artifact is a situation awareness channel, which considers how people are kept informed about what is happening [22]. For Agile teams, the support of different visualization techniques and tools throughout the software development process is crucial to foster awareness and communication. Additionally, this factor can have a positive impact on the sense of purpose [8] when the professionals follow the evolution of their actions within a cycle of continuous improvement.

As a contribution of this paper, we present the methods used for formative evaluation of the artifact. As a result, we found that the SA Canvas is a useful artifact, but some participants reported difficulties in learning and understanding how to use it. To investigate what could be improved, we carried out a participatory design session, where the same subjects pointed out the components of the canvas that needed to be redefined and provided a sketch with a new layout of the canvas' blocks. These results will help us to refine the proposed artifact. In future work, we will empirically investigate the use of SA Canvas in an industrial setting and experienced agile teams in order to get more relevant research results concerning its effectiveness and impact on the software analytics process.

ACKNOWLEDGMENT

This work is financed by National Funds through the Portuguese funding agency FCT – Fundação para a Ciência e a Tecnologia – within project: UID/EEA/50014/2019.

REFERENCES

- [1] T. H. Davenport, "Make better decisions," *Harvard business review*, vol. 87, no. 11, pp. 117–123, 2009.
- [2] R. P. Buse and T. Zimmermann, "Analytics for software development," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 77–80.
- [3] D. Zhang, Y. Dang, J.-G. Lou, S. Han, H. Zhang, and T. Xie, "Software analytics as a learning case in practice: Approaches and experiences," in *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*. ACM, 2011, pp. 55–58.
- [4] R. Robbes, R. Vidal, and M. C. Bastarrica, "Are software analytics efforts worthwhile for small companies? the case of amisoft," *IEEE software*, vol. 30, no. 5, 2013.
- [5] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries et al., "Manifesto for agile software development," 2001.
- [6] A. E. Hassan and T. Xie, "Software intelligence: the future of mining software engineering data," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 161–166.
- [7] D. Hartmann and R. Dymond, "Appropriate agile measurement: using metrics and diagnostics to deliver business value," in *Agile Conference, 2006*. IEEE, 2006, pp. 6–pp.
- [8] O. Liechti, J. Pasquier, and R. Reis, "Supporting agile teams with a test analytics platform: a case study," in *Proceedings of the 12th International Workshop on Automation of Software Testing*. IEEE Press, 2017, pp. 9–15.
- [9] J. Choma, E. M. Guerra, and T. S. Silva, "Patterns for implementing software analytics in development teams," in *Proceedings of the 24th Conference on Pattern Languages of Programs*. ACM, 2017, p. 12.
- [10] —, "Learning from experiments, define quality standards, suspend measurement: Three patterns in a software analytics pattern language," in *Proceedings of the 12th Latin American Conference on Pattern Languages of Programs (SLPLoP)*. ACM, 2018, p. 10.
- [11] D. Zhang, S. Han, Y. Dang, J.-G. Lou, H. Zhang, and T. Xie, "Software analytics in practice," *IEEE software*, vol. 30, no. 5, pp. 30–37, 2013.
- [12] F. Shull, "Data, data everywhere..." *IEEE software*, no. 5, pp. 4–7, 2014.
- [13] J. Guo, M. Rahimi, J. Cleland-Huang, A. Rasin, J. H. Hayes, and M. Vierhauser, "Cold-start software analytics," in *Proceedings of the 13th International Workshop on Mining Software Repositories*. ACM, 2016, pp. 142–153.
- [14] S. Pachidi, M. Spruit, and I. Van De Weerd, "Understanding users' behavior with software operation data mining," *Computers in Human Behavior*, vol. 30, pp. 583–594, 2014.
- [15] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, and S. Ganguly, "Leveraging the crowd: how 48,000 users helped improve lync performance," *IEEE software*, vol. 30, no. 4, pp. 38–45, 2013.
- [16] A. Hevner and S. Chatterjee, "Design science research in information systems," in *Design research in information systems*. Springer, 2010, pp. 9–22.
- [17] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS quarterly*, pp. 319–340, 1989.
- [18] S. Bødker, K. Grønbaek, and M. Kyng, "Cooperative design: techniques and experiences from the scandinavian scene," in *Readings in Human-Computer Interaction*. Elsevier, 1995, pp. 215–224.
- [19] A. Osterwalder and Y. Pigneur, *Business model generation: a handbook for visionaries, game changers, and challengers*. John Wiley & Sons, 2010.
- [20] V. R. Basili, G. Caldiera, and D. H. Rombach, "The goal question metrics approach," *Encyclopedia of Software Engineering*, vol. 1, pp. 528–532, 1994.
- [21] K. Beck and C. Andres, "Extreme programming explained: Embrace change," 2004.
- [22] E. Berndt, D. Furniss, and A. Blandford, "Learning contextual inquiry and distributed cognition: a case study on technology use in anaesthesia," *Cognition, Technology & Work*, vol. 17, no. 3, pp. 431–449, 2015.

Sprint Performance Forecasts in Agile Software Development

The Effect of Futurespectives on Team-Driven Dynamics

Fabian Kortum*, Jil Klünder*, Wasja Brunotte*, and Kurt Schneider*

*Software Engineering Group

Leibniz University Hannover, Germany

Email:{fabian.kortum, jil.kluender, wasja.brunotte, kurt.schneider}@inf.uni-hannover.de

Abstract—In agile software development, the sprint performances and dynamics of teams often imply tendencies for the success of a project. Post mortem strategies, e.g., retrospectives help the team to report and share individually gained experiences (positives and negatives) from previous sprints, and enable them to use these experiences for future sprint planning. The interpretation of effects on sprint performance is often subjective, especially with concern to social-driven factors in teams. Involving strategies from predictive analytics in sprint retrospectives could reduce potential interpretation gaps of dynamics, and enhance the pre-knowledge, also awareness situation when preparing for the next sprint. In a case study involving 15 software projects with a total of 130 involved undergraduate students, we investigated the post-effects on team performances and behavioral-driven factors when providing predictive analytics in retrospectives. Besides measures for productivity, we consider human factors, e.g., team structures, communication, meetings and mood affects in teams as well as project success metrics. We developed a unique JIRA plugin called *ProDynamics* that collects performance information from projects and derives trend-insights for next sprints. The *ProDynamics* plugin enables the use of a times series and neural network model within a JIRA system to interpret factorial dependencies and behavioral pattern, thus to show the next sprint course of a team.

Index Terms—team dynamics, human factors, data analytics, futurespectives, sprint performances, agile

I. INTRODUCTION

In agile software development, accurate sprint estimations and development performances by teams are essential and can cause the difference whether project goals will be sufficiently completed in time, or neglected [1] [2]. However, studies with the focus on software process improvement are often motivated by increasing the overall success of projects, with the help of using modern methods or technologies that enable additional feedback [3] [4]. For organizations, sprint feedback is important and valuable because it allows the teams to receive both subjective and objective information from different perspectives, e.g., productivity monitoring systems like JIRA, customer responses, and team perceptions also experiences. The feedback mentioned above strongly relies on human factors [5]. While tool-oriented technologies and development frameworks are continuously improved, do the social-driven factors of the team present particular difficulty

in the development process chain. Accordingly, the need for understanding the effects of human factors has continuously grown in relevance for the software engineering discipline, in particular for the improvement of the development process [6] [7]. Towards this, team feedback with focus on past sprints conditions can provide a substantial insight towards earlier experiences, dysfunctional habits or positive performance influences. For example, feedback through retrospective sprint characterization and visualizations benefits of post-mortem summaries, similarly to sprint reports [8]. However, dependency implications or trend highlighting concerning team behavior or performances changes over time are barely considered or hard to grasp. This leads to possible knowledge and interpretation gaps when preparing follow-up sprints because previous effects may not be fully identified or adequately encountered. It is highly desirable to give teams a more sustainable feedback opportunity that involves sprint feedback involving both, retrospective and future characterization. Subsequently, we derive the following two research questions from the above-reported context.

RQ₁: Do agile teams show affects for their self-organization based on previous sprint highlighting and customer satisfaction feedback?

RQ₂: Can sprint performance forecasts based on retrospective data help teams to improve development performances in follow-up sprints?

We addressed and investigated *RQ1* and *RQ2* within a case study involving 130 undergraduate students working in 15 software projects, all founded from industrial, government or public institutional partners. The teams followed a Scrum-oriented development process with support through the project management software JIRA. The case study involves weekly self-assessments resolved in JIRA. The question set covers team behavioral driven features, to gain the situational dynamics in a project with effect for the performances in sprints — satisfactory reflections of customer became additionally elicited at the end of every sprint. Productivity information, e.g., velocity during sprints, estimation gaps or sprint interventions become tracked and accessed directly by

JIRA. Half of the projects (seven) were granted to obtain a JIRA plugin called *ProDynamics* [8]. The plugin is based on earlier studies, enabling teams to give and receive additional retrospective feedback for a better knowledge transfer at the end of sprints [8] [9]. In this study, we extended the primarily retrospective feedback in *ProDynamics* with two predictive analytics features. Teams with access to *ProDynamics* can additionally perform time series analyzes for short-term forecasts, also sprint performance tendencies with the help of a neural network encoder-decoder model [10].

This paper is structured as follows. In Section II, we discuss related work on team behavioral effects, feedback and data analytics adaption. Section III provides a brief overview of the study context. In Section IV, we address the methodology about self-assessments in sprints, also time series and neural network forecasts. In Section V, we describe and interpret results. Section VI concludes our research and future work.

II. RELATED WORK

This study builds on previous results and related work with the focus on human-centered software engineering, and the relevance of fast feedback in an agile context.

Human-centered software engineering takes an import role in modern software development [5]. The focus on team communication, self-organization, and well-working relationships within teams reflect the need for better understanding of behavior-driven factors [9]. When planning sprints in agile software development, pre-knowledge on people having different personalities, skills, and ambitions within the organizational structure is crucial [7]. By means, understanding such human interdependencies enrich project improvements throughout better team performances due to a reduction of estimation gaps [11]. Besides, e.g., the meeting and communication manner or atmosphere changes over time can grant valuable project directions towards potential follow-up conflicts or misleading habits that endanger the sprint performances [12] [13]. Nevertheless, human-centered software engineering strongly depends on the contribution and self-reflection of teams, in sharing experiences from previous sprints [14].

Whenever targeting sprint performance improvements in agile development, team feedback, and change adaption should be sincerely considered [15] [4]. In the early 2000s, research results by Rising et al. [16] subsequently revealed that team feedback during iterative development phases helps the teams most when also involving customer feedback and reflections. The customer perception forms a substantial base in the improvement process by associating group subjective sprint perceptions with the team and development expectation of stakeholder. Vetrò et al. [3] also focused on the effect of fast feedback cycles in software development. The authors observed the impact of different feedback mechanism when gathering information from software development teams directed affects for the quality and transferability of experiences and knowledge with changes in following sprints.

Retrospectives are commonly applied in Scrum and most other agile processes to share and interpret team experi-

ences on performance effectiveness collected during the last sprint [5] [11]. Knowledge gained this way is then taken into account to estimate the next sprints more effectively according to previous outcomes. However, the interpretation of team-behavioral pattern often remains subjectively. However, computer-supported interpretation of sprint performance becomes increasingly important and visible - study results, e.g., reported by Vetrò et al. [15] show the improvement potentials in combining data analytics with traditional team feedback appraisals to improve future sprint estimations [17] [15]. The authors applied multiple-regression analyzes to characterize behavioral pattern on, e.g., meeting and development manner, to grant teams better insights on factorial influences over time, especially for efficiency hazards occurred during sprints. Our case study combines feedback mechanisms and predictive analytics about human-centered sprint performance factors.

III. CONTEXT OF COMPARATIVE CASE STUDY

This approach focuses on the effects of feedback on development performance by teams when providing additional feedback that involves forecasts and sprint tendencies. The futurespectives considered in this study base on computer-supported analyzes that relates to team and customer reflections. Predictive analytics is also integrated to characterize interdependencies of behavior pattern. We analyzed *RQ1* and *RQ2* using a comparative study design [18] by observing 130 undergraduate students working in fifteen project teams (eight to nine student per team). Each project was pre-estimated with an approximated development complexity of 2,000 working hours, equally distributed over 15 weeks. Seven of the fifteen teams used the *ProDynamics* JIRA plugin, enabling them to characterize the previous team and development performances, and to esteem next sprints with the support of times series forecasts and neural network prediction models.

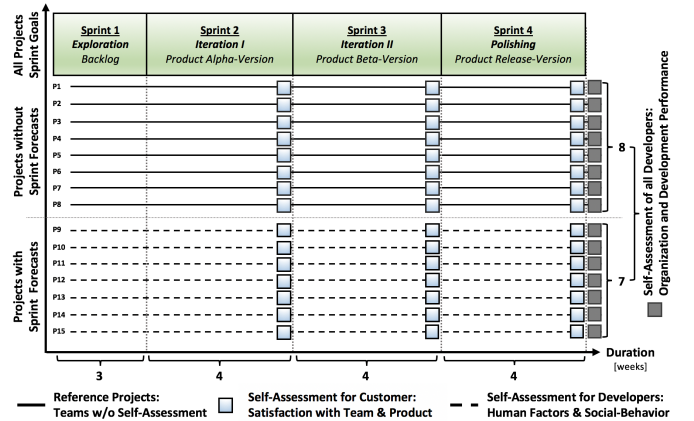


Fig. 1. Comparative Study: Projects, Sprints and Self-Assessments

Figure 1 shows a process overview of the comparative case study design. Participating in the software projects is eligible for students in the fifth semester of their computer science undergraduate studies. The main focus is to collect practical team experience following an iterative development process

including agile methods and practices, such as weekly scrums and storyboards. Students do not receive gradings for participating or performances during the projects. Nevertheless, the course is compulsory within the syllabus. As mentioned before, Scrum was mandatory in each project. All teams had to self-organize inner structures, meetings, and communication, and to manage sprint tasks on a Scrum-board using the project management software JIRA.

For a primary information flow and status exchange, each team was requested to meet face-to-face for at least once a week. For product progress updates, a subsequent meeting with the customer was regularly scheduled once a week. During the sprints, change request could occur, which led to adjusted or discarded issues. At the end of each sprint, teams used retrospectives to highlight and reflect on positive and negative sprint situations. This has mostly been experiences that help the team to better estimate the next sprint tasks and organizational structures. During the projects, additional customer-reflections and satisfaction feedback were assessed to monitor group and development performances also from customer perspectives.

IV. METHODOLOGY

This methodology section starts with (A) the ideology of our JIRA plugin for advanced sprint retrospective and future trend support. The chapter covers the role and realization of (B) the self-assessments on teams and customer satisfaction that were collected as part of the case study. Furthermore, we show how the elicited team and sprint information is later used within (C) the time series and neural network prediction models. In the last subsection, we describe (D) the monitoring process for the sprint performances with a focus on *RQ1* and *RQ2*.

A. ProDynamics –Retrospectives & Futurespective Support:

The project management system JIRA is worldwide known for its team-oriented sprint planning and issue tracking support, commonly used in agile software development. Its usage is scalable from large industrial projects to small entrepreneur solutions. The standard features of JIRA provide substantial help for teams to monitor not yet completed sprints and to derive performance reports for past sprints (development velocity). Lessons learned or experiences gained during a sprint are often reflected by the teams through post-mortem retrospectives [19]. This way, dysfunctional or beneficial sprint characteristics, as well as other individually gained insights, become shared and discussed within the team, enabling them to plan the next sprint in addition to pre-experienced situations or manners. However, reasons for performance affects are not always easy to explain, in particular since the standard JIRA system only characterizes productivity statistics without further implication. We addressed this problem in enabling teams to access fast feedback through a JIRA plugin called *ProDynamics* [8]. In Fig. 2, we give an overview on the sprint analysis features currently provided in *ProDynamics*.

The red box highlights the two newly integrated sprint forecasting features of this study. The plugin addresses the ideology to support and increase the team's understanding and awareness for factorial effects on team-driven sprint behavior and development performance. Previous study results have shown that after suitable preparation of retrospective data, teams can be supported by retrospective computer-aided feedback [8] [15]. However, we believe that the support for such teams in their sprint estimations can further enrich the organizational and development performances with the help of integrated data analytics solutions.

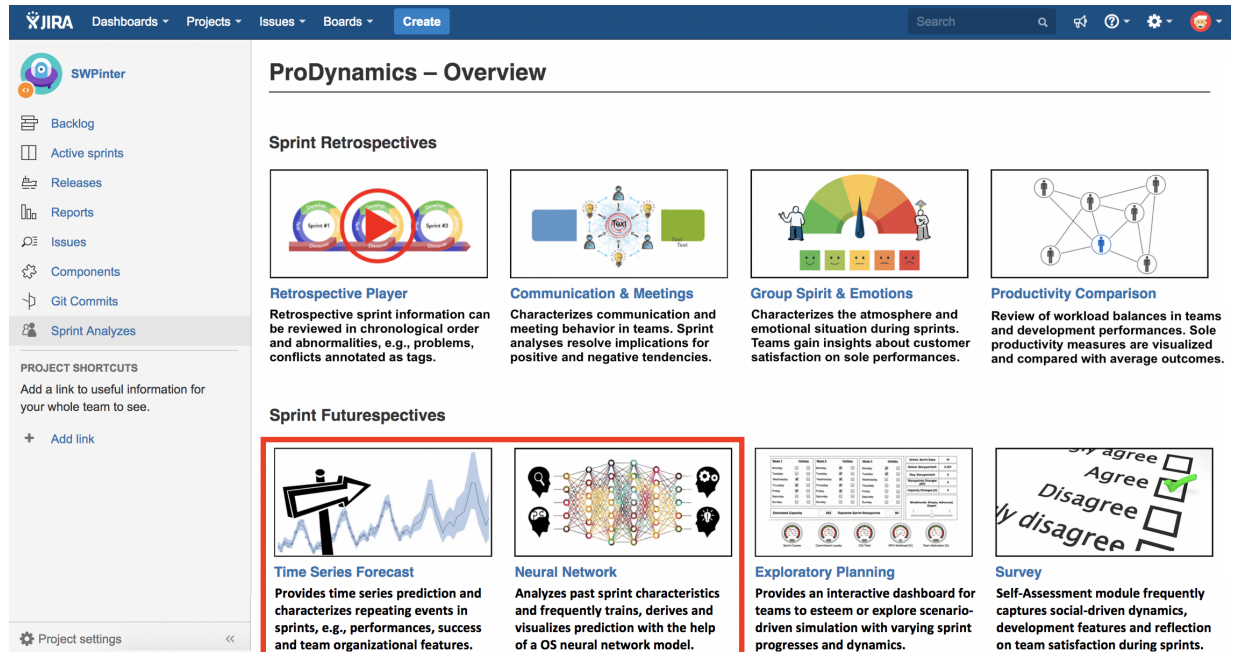


Fig. 2. Behavior-Driven Feedback Support on Sprint Futurespectives in JIRA

B. Self-Assessments for Teams, and Customer-Satisfaction:

Interpretations of previous sprint dynamics or estimations of follow-up sprints concerning the social nature of agile teams can be challenging [7]. Various human factors, such as mood, communication or meeting manner have a direct effect for the ongoing project, while dysfunctional behavior often remains undetected or hard to grasp until problems enlarge [9] [20].

Our approach is designed for teams with an open mentality for self-reflection in exchange for sustainable feedback that enables opportunities for change-driven improvements of organizational and development structures [14] [4]. Using the *ProDynamics* plugin, we enable an integrated self-assessment solution for a systematic elicitation of team dynamics in ongoing sprints. During our comparative case study with $n = 15$ projects, three different assessments were applied to grasp the maximal descriptive team characteristics over time. The assessment designs and question features are based on previous studies, also related work [21] [9], and continuously refined to reach the currently applied versions. The question set is self-adapting, e.g., the information elicitation about communication or meeting behavior only occur for members with active information exchange. With this, the interviewees' effort to complete a survey could be minimized to a length of 1-2 minutes. All assessments are realized through 5-points Likert scales, determining his or her level of agreement on a symmetric agree-disagree scale with predefined sprint or team behavioral statements. In the following, we explain the three assessment types in detail. A summary of the variants, in particular, the intervals, self-assessment sprint information by categories, and interviewees are shown in Fig. 3.

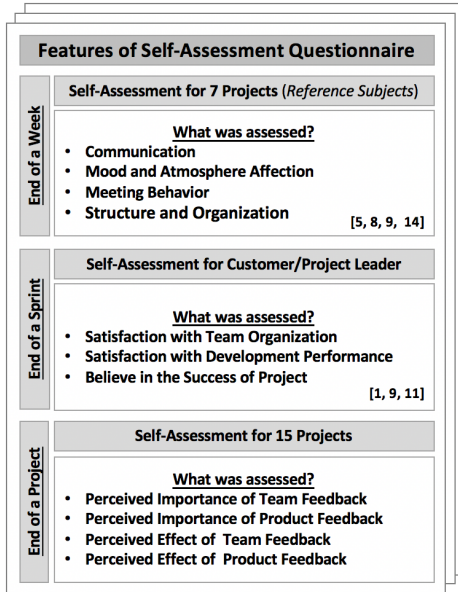


Fig. 3. Self-Assessment Intervals, Interviewees and Features

1) *Self-Assessment at the End of each Week:* During the 15 weeks of this case study, seven out of the fifteen student developer teams voluntarily participated in the study.

Besides the access to the *ProDynamics* retrospective and futurespective sprint features, the participation included a weekly self-assessment for each team member. The assessments capture social- and organization features, such as:

- who-to-whom communication and media channel use
- meeting quantity and average duration
- the atmosphere in the group, personal mood
- satisfaction about the last weeks' performances

We derived other team and sprint metrics, e.g., meeting participation, maverick trends, and centralized communication structures from each team member's response. A summary of all assessment information is shown in Fig. 3. Subsequently, all category features that are currently considered by the time series and neural network model, including productivity measures from JIRA, become listed in Fig. 5.

2) *Self-Assessment at the End of each Sprint:* The second self-assessment question set on the satisfaction with the team and development performances for all 15 software projects was also answered by the customer, and the scrum master at the end of each sprint (except the first). We activated the performance assessments with the second sprint, because the first three weeks of the project were mainly for exploration, to create and fill the backlog, form team structures, get to know the customer and reach a steady state before the next sprint.

However, the survey covers in total ten questions, four about the team organization, four on the development performance as well as two items focusing on the overall satisfaction with the team and product. The question structure became realized through 5-points Likert scales, determining the customers and scrum master's level of agreement on a symmetric agree-disagree scale, similar to the other two self-assessments. The question set was used as one reference indicator between team-driven dynamics during the sprints and potential effects for the customers' satisfaction. The scrum masters' responses become utilized to determine possible offsets between external views and the team inside knowledge. For the comparison of satisfaction changes between the teams with access to *ProDynamics* and those that did not, the customer and scrum master of all fifteen projects were invited to complete this self-assessment form.

3) *Self-Assessment at the End of each Project:* The third self-assessment took place only once at the end of each project and became applied to all 130-student developer. The survey includes questions on the personal perception of the developers, e.g., whether there were moments with a need for additional feedback during the sprints, and if there were recognizable effects (positive or negative) within the own team in case of provided feedback. The following four assessment features became only ones elicit at the end of each project for the validation of every student's perception of their team and development performances during the 15 weeks:

- importance of organizational feedback for the team
- importance of product feedback for the development
- perceived effect in the team because of team feedback
- perceived effect in the team because of product feedback

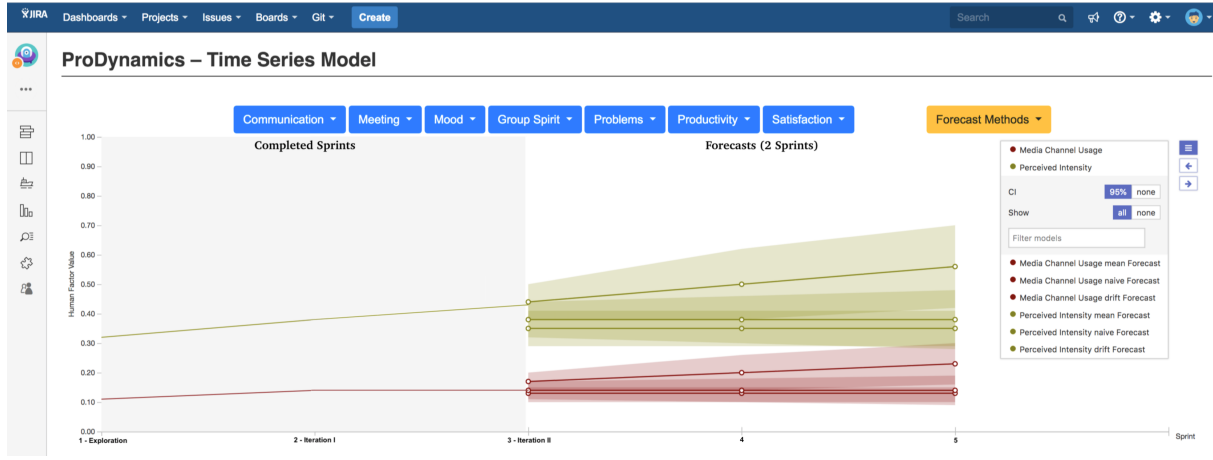


Fig. 4. Time Series Sprint Forecasts derived in *ProDynamics*

The responses were compared with the customer and scrum master satisfaction after each sprint. This survey also involves questions about the need and usefulness, e.g., of a centralized feedback solution in JIRA. Also, whether JIRA was an adequate solution to manage and organize sprints during the case study project. Those are by-product information, in particular with no relevance to answer *RQ1* and *RQ2*.

C. Sprint-Trends through Time Series and Neural Networks

With this case study, we investigated whether teams can gain a more sustainable use of feedback when considering both, past retrospective records and future sprint performance predictions. Besides this, we incorporated retrospective sprint characteristics (e.g., organization structures, communication- and meeting behavior, productivities, motivation) with two predictive models to grant teams an additional trend characterization on behavior-driven factors when esteeming the next sprints. We chose both predictive methods based on their functional properties in supporting inference-statistical analysis, e.g., sprint series. The analysis helps to estimate sprint and team measurements in the future based on past team-behavior in sprint sequences.

The times series forecast used for the sprint-trend esteems based on an open source java library published by Workday¹. The library provides time series analyzes, involving ARIMA-, Mean-, Naïve- and Drift-forecasts. The forecasting model in this study become fitted by the measurements listed in Fig. 4.

The ARIMA-model characterizes seasonal inferences from past and forecasts future points in the series [10]. The Mean-method derives the arithmetic mean from the sequence of sprint data to esteem follow-up values within fitted parameter ranges. The seasonal Naïve-method uses sprint metrics from the second sprint week on to predict the third sprint parameters. The prediction of the fourth sprint metrics is derived with the values from the third sprint, and so on. The Drift-method obtains a straight line between the first and last data point to characterize the sprint metrics tendency drift. Seasonal patterns

are not taken into account with this Drift-method. Figure 4 shows a time series forecast for the communication metrics Media Channel Usage and Perceived Intensity.

The time series viewer enables the teams to choose from one to four supported forecasting methods. The interactive chart allows a user to select different past sprints and the underlying metrics. With this, the chosen sprint metric(s) become analyzed and forecast with the help of the four forecasting methods. The prediction interval can reach a maximal length smaller than the number of yet completed sprints. The colored lines within the gray background area shown in Fig. 4 present the real data points for two selected communication parameters. The colored areas on the right half of the chart mark the 95% confidence interval of the prediction of each forecast. For example, the maximally available forecast horizon in Fig. 4 is two sprints, because the time series model derives its prediction based on three yet completed sprints. Sprint forecasts are labeled on the time axis through a counter and completed sprints name tags.

The *ProDynamics* – Neural Network viewer focuses on the second prediction approach for estimating social-driven team measurements based on retrospective sprint and team records. The neural network is implemented in using the open source library Deeplearning4j. The Deeplearning4j-library covers cross-platform algorithms on machine learning and artificial intelligence is implemented in Java and runs in a JVM such as used by the JIRA system. Similar to our time series forecast solution the neural network viewer provides past sprint metric plots based on the user selections.

Also, the neural network model allows the team to review past sprint conditions covering organizational structures, social-driven team behavior, customer satisfaction, productivity measures as well as problem and conflict appearances. An overview of all covered factors is listed in Fig. 5, which is a result of the assessed team and customer responses as well of development performances that is natively tracked in JIRA. The model is trained with the listed data features considering the availability of already completed sprint records.

¹ <https://github.com/Workday/timeseries-forecast>

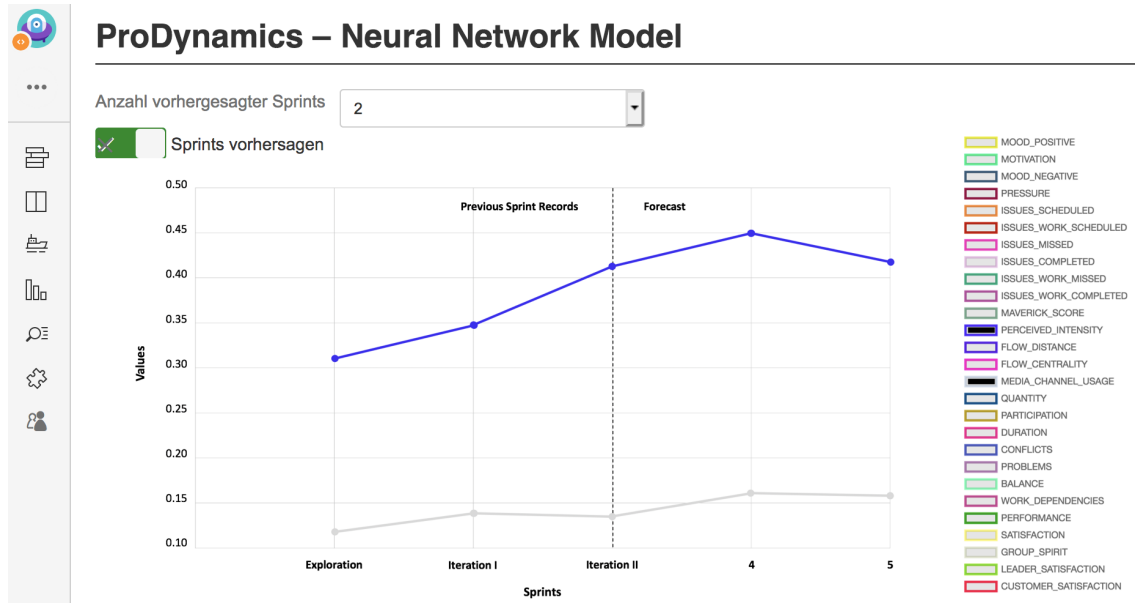


Fig. 5. Neural Network Sprint Predictions derived in *ProDynamics*

Training the model requires enlarged computation effort for the data encoding process. Therefore, model updates are performed automatically, but only once during the weekend. The neural network viewer enables the teams to choose individual real data plots of past sprints, or with an active future horizon. The maximal size of future-horizon is limited to the number of yet completed sprints minus one, similar to the time series model. The interactive chart allows a user to select between the sprint metrics and plot trends for the considered team-driven factors. The colored lines in front of the dash border line present the real data points for the two chosen communication parameter Media Channel Usage and Perceived Intensity. The colored lines on the left side of the dashed border present the computed prediction according to the feature records encoding of three yet completed sprints.

D. Sprint Performance Monitoring

In this study, the sprint performances of teams' base on the development (velocity of tasks) as well on the organizational performances during each sprint. The velocities in all teams are comparable productivity measures tracked within JIRA, thus did not require to be separately assessed. This enabled us a direct performance comparison was between a particular group and the average of all other teams during a sprint. Besides the sole productivity measures of teams, we used the customer and scrum master satisfaction feedback after each sprint to determine whether the development outcome also fulfilled product expectations according to quality and functional requirements.

However, the team performances over time with concern on social-driven changes due to futurespective feedback became solely traced through the customer, and scrum master feedback elicited at the end of every sprint. We are reasoning this processing with the fact, that only half of all teams could

access the *ProDynamics* futurespectives, and also frequently completed the self-assessments on social-driven behaviors. In considering the customer and scrum master perceived team performances during each sprint, we could compare the organization performance changes of all teams. Subsequently, the effects and trends could become characterized due to the comparative study subjects with different sprint estimation and planning support. Of course, at this appraisal level, sole factorial influences become not closer taken into account. However, it allowed us first interpretations about whether teams adopt the *ProDynamics* usage, also whether groups with access get used to derive better sprint estimations with constant or even positively improving customer satisfaction outcome, on organizational and product aspects.

V. INTERPRETATION AND VALIDITY OF RESULTS

In the following two subsections, we statistically interpret and discuss the effects of *ProDynamics* futurespectives on the sprint performances, also emphasize related threats to validity.

A. Interpretation of statistical measures

In Section IV, we described the sprint performances as a result of sprint productivity (velocity), also the customer and scrum master satisfaction responses on the team and product performances after every sprint. With the help of Pearson correlation analysis and 2-paired t-Tests, we determined sprint performance differences between the seven groups that actively used the *ProDynamics* prediction features and the other eight teams without access. We found out that the teams with access to the provided sprint forecasts showed fewer estimation errors, therefore with more optimal velocity distribution at 98% as the comparison of the orange boxes (1) in Fig. 6 and Fig. 7 reveal. The overall sprint estimation error for those teams was $\pm 9\%$. In particular, do the groups

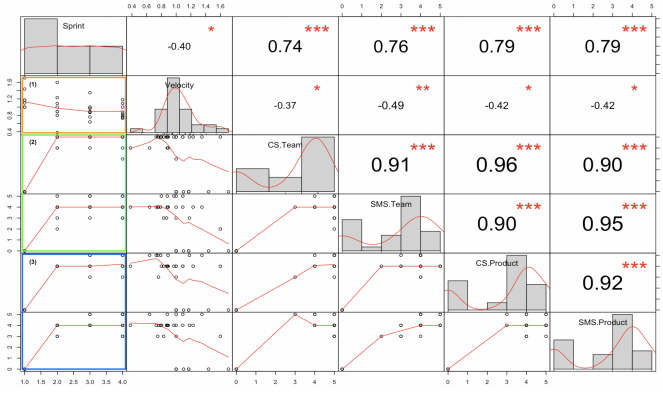


Fig. 6. Team Projects without *ProDynamics*-Futurespective access

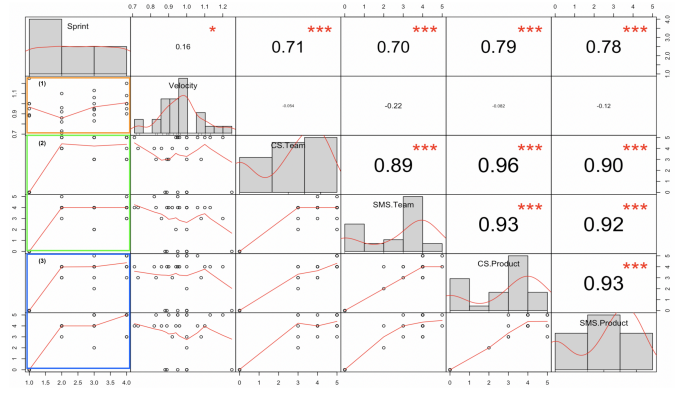


Fig. 7. Team Projects with *ProDynamics*-Futurespective access

without access to *ProDynamics* showed strong tendencies for over-estimating their sprint tasks during the first two sprints, followed by underestimations that caused strong deviations between the number of scheduled tasks and completed ones. Due to this, an overall sprint estimation error of $\pm 19\%$ was identified. However, in comparing the yellow markings (2) in both figures, the team's organizational performances revealed no benefits given due to the additionally provided sprint forecasts. Moreover, the outcome is on a constant level, while both, customer and scrum master reflected a sustainable organization and communication structure in most teams.

The third factor of the sprint performances involves the software product, in particular, the quality, and requirements fulfillment after each sprint. The futurespective in *ProDynamics* enabled a few teams to preview customer satisfaction according to previous performances. However, the forecasts only highlight chances for adjustment, while the teams decide whether to use the available information to improve the previous situation. By comparing the blue boxes (3) in Fig. 6 and Fig. 7, an affect for teams with *ProDynamics* usages becomes reflected throughout an increasing product satisfaction by the customer and scrum master. The satisfaction increase can be of course also because of excitement about the product majority.

Nevertheless, a significant rise in teams with customer feedback knowledge could be measured, while comparable projects remained on a constant level. For redundancy, we also considered the product satisfaction of scrum master, which significantly correlates with our interpretation. Beside results of this case study showed strong accordance between customer and scrum master perceived team and development performances. While the scrum master usually tends to have more team internal information and critical knowledge about accomplishments, does the deviation with external customer ratings present a low perception gap.

B. Threats to validity

Construct validity: We looked at the social-driven team affects only through statistical and artificial methods. The sprint information features obtained in *ProDynamics* are chosen based on previous studies [18] [13]. However, the ac-

curacy of the predictions strongly depends on the quality and completeness of the self-assessed data. The effects on performances depend on whether the teams considered the sprint forecasts in follow-up esteems. Customer satisfaction responses as one success indicator might have involved offsets, because of unjustified expectation or lack of experience.

Internal validity: The interpretations and results of this study rely on the self-assessed team and customer feedback. Only voluntary groups completed surveys, in exchange for accessing additional prediction support in JIRA. Therefore, team responses can be assumed to be accurate and unbiased [16]. Participating teams accepted the privacy limitation, e.g., communications and performances were viewable, but exclusively by the assigned team.

External validity: Since we examine student teams, the results should not be overgeneralized. However, the software projects are founded by a real customer from industry, public institutions or governments. Hence, data collected from other company or university projects could lead to different results. Due to further limitations like the involvement of social-driven factors and unknown domain influences, our interpretations may not embrace all possibly existing project scenarios.

Conclusion validity: All interpretations are plausible and statistically valid. However, there may be different self-assessment responses when repeating the project with the same participants: team-behavioral factors, emotions, skills or unknown influences could have changed in the meantime. Subsequently, the accuracy of predictions could vary due to differently completed surveys, and project progresses. However, the methodology can be generalized and applied to various agile projects that allow assessments on team and sprint information.

VI. CONCLUSION

This study focuses on the effects of social-driven dynamics in agile software developments when providing teams additional feedback on sprint tendencies. To determine the impact of futurespective feedback, we realized a JIRA plugin named *ProDynamics* that simplifies the elicitation of team-driven factors as well as performance measures within JIRA.

The feedback becomes resolved through sprint series forecasts and neural network predictions in an integrated JIRA plugin solution. The computer-based sprint analyzes use team and customer reflections that were frequently assessed, analyzed and characterized for factorial interdependencies on development performances and team-driven behavioral pattern.

With the help of a comparative case study involving fifteen software projects with 130 students, throughout 15 weeks, we gathered weekly information about communication, meeting and emotional metrics from half of the projects. The elicited data became frequently used to train time series and neural network models, enabled the 7 out of 15 groups to gain additional insights about previous sprint and team performances, also derive trend-forecasts for follow-up sprints. Measuring the performance differences between the groups with pro-active feedback and those without involved customer and scrum master feedback from all 15 projects, that became repeatedly elicited at the end of every sprint. The feedback covered past sprint perceiving on both, team and development performances.

Pearson correlation statistics helped us to interpret the effects on sprint performance, in particular, the team and development performance in each project. We found statistical evidence towards that the groups with access to the additionally provided *ProDynamics* forecasts showed a definite decrease for sprint estimation gaps by 10%, while the groups without *ProDynamics* access tend to have more volatile velocity performances. We could show, that the additional use of forecasting methods supports the groups to interpret customer satisfaction better, thus improve the product outcome at the end of sprint. The study also revealed, that teams not necessarily adjust internal organization structures due to predictive information. Most of the groups showed an almost steady level in their weekly communication and meeting behavior, towards no significant affects could be determined.

We can conclude that the *ProDynamics* futurespectives enabled a sustainable team organizational and development performance improvement for the groups with access to the plugin. The team performances dynamics during the sprint sequences showed strong stabilizing characteristics, due to more accurate sprint estimates compared with the comparison groups that only used general sprint information in JIRA, e.g., burndown- and velocity charts. Besides, the *ProDynamics* plugin realized a simplified data elicitation for social-driven team factors, while some group had could reach a positive effect for follow-up sprint executions.

We are currently working on a newer version for the *ProDynamics* plugin, that does extend the retrospective, and futurespective sprint analyzes, by a planning-oriented simulation feature. With this, we believe that teams can potentially improve sprint estimations because of training effects. Various scenarios could be explored before an official sprint start, by incorporating past performances with a generalized simulation model for agile development processes, e.g., system dynamics. A simulation-based approach could grant the team a better insight about appearing behavioral dynamics over time, also help to discover new characteristics, that would remain

undetermined otherwise. Generally spoken, simulations could gain further knowledge and train the sprint estimation skills of teams and project manager.

ACKNOWLEDGMENT

This work was funded by the German Research Society (DFG) under the project name Team Dynamics (2018-2020). Grant number 263807701.

REFERENCES

- [1] N. Agarwal and U. Rathod, "Defining 'success' for software projects: An exploratory revelation," *International journal of project management*, vol. 24, no. 4, pp. 358–370, 2006.
- [2] P. Mohagheghi and M. Jørgensen, "What contributes to the success of it projects? an empirical study of it projects in the norwegian public sector," *JSW*, vol. 12, no. 9, pp. 751–758, 2017.
- [3] A. Vetrò, S. Ognawala, D. M. Fernández, and S. Wagner, "Fast feedback cycles in empirical software engineering research," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2*. IEEE Press, 2015, pp. 583–586.
- [4] L. Williams and A. Cockburn, "Agile software development: it's about feedback and change," *IEEE Computer*, vol. 36, no. 6, pp. 39–43, 2003.
- [5] A. Cockburn and J. Highsmith, "Agile software development: The people factor," *Computer*, no. 11, pp. 131–133, 2001.
- [6] F. Kortum, J. Klünder, and K. Schneider, "Characterizing relationships for system dynamics models supported by exploratory data analysis," in *29th International Conference on Software Engineering and Knowledge Engineering. KSI Research Inc.*, vol. 15, 2017, pp. 39–43.
- [7] E. Whitworth and R. Biddle, "The social nature of agile teams," in *Agile 2007 (AGILE 2007)*. IEEE, 2007, pp. 26–36.
- [8] F. Kortum, J. Klünder, and K. Schneider, "Behavior-driven dynamics in agile development: The effect of fast feedback on teams," in *Proceedings of the 2019 International Conference on Software and System Process*. ACM, 2019.
- [9] K. Schneider, O. Liskin, H. Paulsen, and S. Kauffeld, "Media, mood, and meetings: Related to project success?" *ACM Transactions on Computing Education (TOCE)*, vol. 15, no. 4, p. 21, 2015.
- [10] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.
- [11] N. B. Moe and T. Dingsøyr, "Scrum and team effectiveness: Theory and practice," in *International Conference on Agile Processes and Extreme Programming in Software Engineering*. Springer, 2008, pp. 11–20.
- [12] S. Kauffeld and N. Lehmann-Willenbrock, "Meetings matter: Effects of team meetings on team and organizational success," *Small Group Research*, vol. 43, no. 2, pp. 130–158, 2012.
- [13] J. Klünder, K. Schneider, F. Kortum, J. Straube, L. Handke, and S. Kauffeld, "Communication in teams-an expression of social conflicts," in *Human-Centered and Error-Resilient Systems Development*. Springer, 2016, pp. 111–129.
- [14] C. J. Stettina and W. Heijstek, "Five agile factors: Helping self-management to self-reflect," in *European Conference on Software Process Improvement*. Springer, 2011, pp. 84–96.
- [15] A. Vetro, R. Dürre, M. Conoscenti, D. M. Fernández, and M. Jørgensen, "Combining data analytics with team feedback to improve the estimation process in agile software development," *Foundations of Computing and Decision Sciences*, vol. 43, no. 4, pp. 305–334, 2018.
- [16] L. Rising and N. S. Janoff, "The scrum software development process for small teams," *IEEE software*, vol. 17, no. 4, pp. 26–32, 2000.
- [17] F. A. Batarseh and A. J. Gonzalez, "Predicting failures in agile software development through data analytics," *Software Quality Journal*, vol. 26, no. 1, pp. 49–66, 2018.
- [18] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and software technology*, vol. 50, no. 9–10, pp. 833–859, 2008.
- [19] J. Highsmith and A. Cockburn, "Agile software development: The business of innovation," *Computer*, vol. 34, no. 9, pp. 120–127, 2001.
- [20] K. Schneider, J. Klünder, F. Kortum, L. Handke, J. Straube, and S. Kauffeld, "Positive affect through interactions in meetings: The role of proactive and supportive statements," *Journal of Systems and Software*, vol. 143, pp. 59–70, 2018.
- [21] J. A. Ross, "The reliability, validity, and utility of self-assessment," 2006.

Research on Multi-constraint Combinatorial Test Technology for High Confidence Embedded Software

Feng Gao ,Fei Deng,Yunqiang Yan
Institute of Computer Application
China Academy of Engineering Physics
MianYang,China
achang85@163.com

Abstract—The importance and complexity of software in embedded devices are increasing. It becomes an active research topic on how to carry out the full and efficient testing of military high confidence embedded software such as weaponry, aerospace and so on. The combinatorial test is an effective test case generation technique. But the complexity of the constructor of the combinatorial test suites is NP-complete. The effectiveness and complexity of combinatorial test methods have attracted researchers in the field of combined mathematics and software engineering to study it deeply. For the characteristics of military embedded software, a multi-constraint combinatorial test method is proposed. This method takes into consideration the constraint relationship among parameters, and uses it to guarantee the covering of the seed combination and the simplification of the use case set. And it implements the test case generation tool based on this method. The results show that this tool generates a few use cases and can guarantee the covering of key combinations.

Key Words: *High Confidence; Embedded Software; Multi-constraint; Combinatorial Test*

I. INTRODUCTION

With the development of information technology, the importance and complexity of embedded system are higher and higher. The software caused by the embedded system fails covers about 70%[1-2];Software becomes an important factor restricting the quality of embedded system. Software testing is an important means to ensure software quality. It becomes an active research topic on how to carry out the verification and verification (V&V) of high - efficient and high - quality weapon equipment, aerospace and other high confidence embedded software[3-6]. A lot of practice shows that combinatorial testing is an effective software testing method. It generates a small amount of high-quality test data and systematically tests the combination of parameters. Testing case set generation is a hot topic in combinatorial testing research. People have presented many mathematical methods, heuristic search methods and various greedy algorithms. But these methods have their own limitations and can only have certain advantages when solving some certain problems. For example, TConfig[6] is a mathematical method to construct the recursive structure by using the orthogonal tables and other basic components. This method has a fast generating speed. But the downside is that it need to rely on existing algebra or combined objects. For the heuristic search, we can use tabu search [7], simulated annealing (SA)[8] and other methods to generate small test

case sets. But these methods generally take a long time. In contrast, heuristic greedy algorithms are not only flexible but also fast. At present, a variety of heuristic greedy algorithms have been presented such as AETG[9-11], etc.TCG[12], DDA[13-14] and IPO[15]. Each of these methods has its own advantages. They are all universal methods which are not ideal for solving some specific problems.

In this paper, according to the characteristics of military high confidence embedded software such as weapon equipment, aerospace and so on, a multi-constraint combination test method which can configure seed combinations, covering strength and parameter constraints is proposed. This method takes into consideration the importance of different parameters and the constraint relation between the parameters, taking advantage of them to ensure the covering of the seed combinations and the simplification of the use case set. And it implements the test case generation tool based on this method. The test result shows that this tool generates a few use cases and can guarantee the covering of key combinations.

The second section introduces the basic concepts and models of combinatorial test. The third section presents the multi-constraint combinatorial test methods. And the fourth section describes the experimental design and the results, finally the summary and outlooks.

II. THE BASIC MODEL OF COMBINATORIAL TEST

Assume n parameters are affecting the system software SUT(software under testing), recorded as $F = \{f_1, f_2, \dots, f_n\}$. These parameters can be SUT configure parameter internal events external input, etc. The parameter f_i has a_i possible values by equivalence partitioning, boundary value method and other pre-parametric decomposition, forming the value set $V_i = \{a_1, a_2, \dots, a_i\}, 1 \leq i \leq n$.

Definition 1. Regard n -tuple (v_1, v_2, \dots, v_n) as a test case for SUT, and $v_1 \in V_1, v_2 \in V_2, \dots, v_n \in V_n$.

Correspondingly, regard a collection of n tuples as a test case set for SUT. In the combinatorial test, the test case set is often referred to as a combination covering table (Covering Array for short).

Definition 2. The t -dimension overlay table of the system SUT $CA(N; t, n, (v_1, v_2, \dots, v_n))$ is an $N \times n$ array. The i column corresponds to the i parameter, which is recorded as v_i . The subarray of $N \times t$ that is formed by any t parameter contains all

DOI reference number: 10.18293/SEKE2019-016

of the t tuples of the t parameter, in which t is the strength of the combinatorial test.

According to the definition of the covering array CA(covering array) which is given by Cohen et al. to describe the set of test cases in a combinatorial test case, generally, an array of overrides that can cover any two parameters of any parameter is called a two-dimensional combination (2-way) or Pairwise combination. t -way is the combination of values that can cover the t parameters. When t is equal to the number of parameters n , the override array can override all of the combined conditions of the parameter system, which is 100% full coverage. Therefore, the mathematical model of the combinatorial test case set is often non-continuous, multi-objective and nonlinear constraint solving the problem.

III. MULTI-CONSTRAINT HCESCT COMBINATORIAL TEST METHOD

The most important difference between military embedded software and general software is that the security and reliability requirements of military embedded software are extremely high, and it is best to have 100% test case coverage. But some special system software that has a combination explosion or an important level of software can't reach 100% complete coverage because of many considerations for military software testing, the complexity of operating environment, time and cost. Reasonable use of combinatorial test technology for the military embedded system software to select effective test case set can make up the randomness of artificial design cases and randomness of test cases.

HCESCT (High Confidence Embedded Software Test Generation Method for Combinatorial Testing) is the combinatorial test case generation method presented by this paper with military embedded software as the test target. In combination with the characteristics of high reliability and security requirements, HCESCT method is mainly concerned with adopting reasonable strategies to solve the problem of explosion of cases in the test process. At the same time, ensure that important values are not missed. The combinatorial test method process in this paper is shown in Fig. 1.

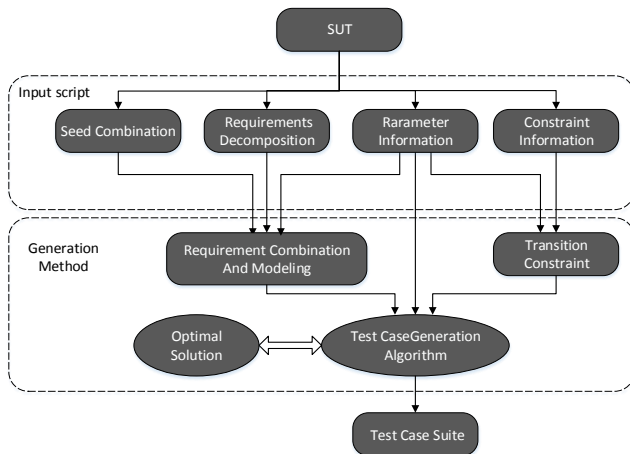


Fig. 1. The Combinatorial Test Method Flow Chart

According to the characteristics of high confidence embedded software, some key parameters must be covered. Depending on the importance of parameters, some require only 2-dimensional coverage, but some combination of parameters need to be covered in higher dimensions. In real software, there is always a certain dependence relation between the parameters of software, which leads to some constraint relationship among some values in these parameters. In order to support the various flexible strategies of the military embedded software combinatorial test method, important combinations must be guaranteed not missing. Here are a few effective generation strategies.

A. Seed Combinations

In high confidence embedded software with high-reliability requirements, it is particularly important to have some parameter combinations. If these parameter combinations go wrong, they can have disastrous consequences. Therefore, the Seed combination (Seed) strategy is added. At the stage of software portfolio testing modeling, add the combination of parameters that must be combined in the software application scenario as a seed combination to ensure that the use case set generated at the end of the HCESCT algorithm is sure to screen out these important combinations. The modeling approach is as follows:

[SEED]

P1:v1,P2:v2,P3:v3;

This constraint statement (P1,P2,P3)=(v1,v2,v3) must be present in the test case

B. Variable Intensity Coverage

Also in order to ensure the coverage of the tested software, the coverage of about 70% of the matched pairs is obviously poor. In multi-parameter systems such as influencing factors, software input and pattern categories, it is necessary to adopt a flexible and variable strategy to ensure the high-dimensional combination of important parameters and the secondary parameter matching combination. For example, on a loaded software, functional requirements specify that the most important parameters affecting software output include the launch mode parameters Mod, temperature Tem, wind power WindF and the wind condition WindD. In parametric system modeling, the variable intensity coverage strategy should be used to cover the four parameters in a 3-way or 4-way combination. At the same time, other parameters with little effect on the launch function such as humidity, launch time and altitude use a matching combination. Enter in the modeling script:

[STRENGTHS]

default : 2; // The global coverage intensity is matched.

Mod, Tim, WindF, WindD : 3; // Variable intensity, important parameter high dimensional combination

With some scripts as input, this algorithm combines the parameters of different important levels with varying intensity.

C. Parameter Constraint

In military software, there're always situations that signal A is received, but parameter b-e fails or signal B is received, but parameter M-N fails. This situation is called a

conflict among parameters. To solve this problem, the parameter constraint description is added to the script during the parameter modeling phase. Adding the parameter constraint strategy, while searching in the generation algorithm, for any parameter:

- Collect all the conditions that block this parameter;
- The parameter is blocked when any condition is satisfied;
- This parameter must be valid when all conditions are not satisfied.

In accordance with the above judgment logic, the algorithm is able to automatically block the combination of invalid parameters that have been identified, and prevent the algorithm from generating too many invalid test cases. The parametric constraint modeling method is as follows:

[CONSTRAINTS]

$P1 == v1 \rightarrow P2 != v2;$

$P3 == v3 \rightarrow P4 == v4 \& P5 > v5。$

D. HCESCT Algorithm Introduction

As mentioned above, HCESCT uses the greedy algorithm that incorporates a flexible and practical combined strategy. Adopt the classic line-by-line search and a one-time-one-line generation strategy:

1) *Initialize covering requirements (Target combination: a combination that needs to be covered)*

- Intensity—combination of all two parameters

2) *Generate test cases per article*

- Try to cover many of the uncovered target combinations.
- The new test case must satisfy all constraints.
- Combinatorial optimization problem

3) *When a new target combination cannot be overridden, stop.*

- The remaining uncovered target combination violates the constraint and cannot be overridden.

The methods of constraint in the algorithm are as follows:

- Convert to a forbidden combination
- The constraint solving technique is adopted to ensure the correctness of the results.

The algorithm framework is shown below:

TABLE I. ALGORITHM KERNEL ALGORITHM OF HCESCT

Algorithm Kernel algorithm of HCESCT
1: init(combination_set);
2: while true do
3: gen_opt_problem();
4: if solve() == OK then
5: new_test_case= translate_solver_output();
6: test_suite.add(new_test_case);
7: update(combination_set, new_test_case);
8: else
9: break ;
10: end if

11: **end while**

The traditional greedy algorithm produces a new test case that determines and overrides the uncovered combination in the while. This is an optimization problem for both search methods and heuristic methods. There are some algorithms for handling parameter constraints. Distribution parameter values in the algorithm, for example, they call external solver or other methods to determine whether the test cases to satisfy the constraints, or to determine which parameter values can be assigned to the parameters in recycling. In contrast, our algorithm integrates generation optimization and constraint solving together. In each while, every time a new test case is generated, only the optimization problem is considered to ensure that the new test case covers at least one uncovered combination. The judgment and retry process that generates the use case satisfies the constraints are done in the external solution function, not in the upper algorithm while. Because the constraint solving function solve the parameter constraint judgment problem very well, this will save a lot of resources in the overall cost of the algorithm. HCESCT algorithm diagram is shown in Fig. 2.

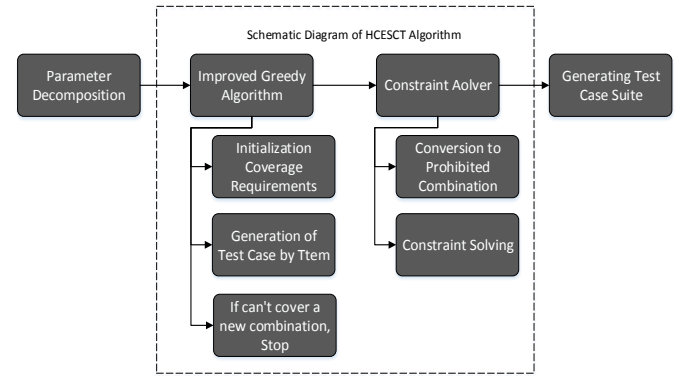


Fig. 2. HCESCT Algorithm Diagram

IV. THE COMBINATORIAL TEST EXPERIMENT AND THE GENERATION EFFECT CONTRAST

To illustrate the method of combinatorial test strategy in this paper and contrast the generation effects of existing main tools, some high confidence embedded data processing software are taken as an example. The software is in the demand analysis phase. And the type of input is shown in the table below:

TABLE II. INPUT DATA TYPE

	A Group P Data	A Group M Data	B Group M Data	B Group M Data
Data Source 1	0	0	0	0
Data Source2	0	0	0	0
Data Source3	0	0	0	0

Data processing software has three different types of data sources, and each data source is independent of each other. The software can accept data from three sources at a time. The data format of each data source is one of the four types of A group P data, A group M data, B group P data and B group M data. Among them, the A group P data of data source 1 and the B group P data of data source 3 is an important combination of data sources and the most frequently processed data of software. Similarly, data source 1 and data source 2 will not send B group P data to the data

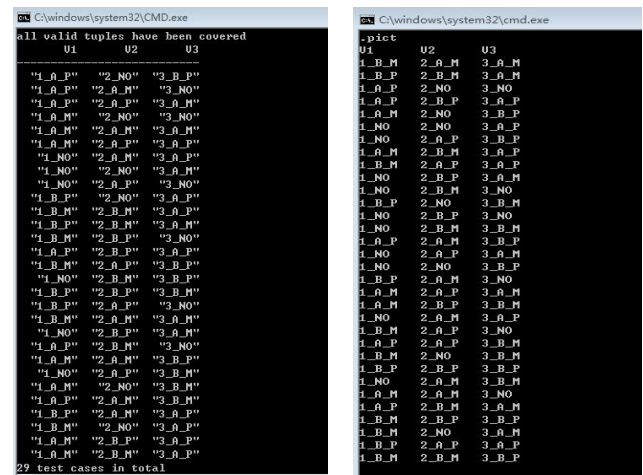
As is shown in HCESCT combinatorial test method flow chart, after this data is decomposed into embedded software requirements, a requirement combination is required. That means taking A group P data of data source 1 as a parameter value. The parametric model after modeling is shown in table 3.

	A Group P Data	A Group M Data	B Group M Data	B Group M Data	No Data
Parameter V1	1_A_P	1_A_M	1_B_P	1_B_M	1_NO
Parameter V2	2_A_P	2_A_M	2_B_P	2_B_M	2_NO
Parameter V3	3_A_P	3_A_M	3_B_P	3_B_M	3_NO

The HCESCT combinatorial test method was used to analyze the demand of the measured parts and integrate and model the requirements. The seed combinations are: V1=1_A_P , V2=2_NO , V3=3_B_P. When the parameter constraint is V3=3_B_M, V2!=2_B_M; When the parameter constraint is V3=3_B_P, V2!=2_B_P , V3!=3_B_P. Use the script as shown in the following diagram to generate input for the combinatorial test case generation algorithm.

```
TestModel-INPUT
[PARAMETERS]
string V1: "1_A_P", "1_A_M", "1_B_P", "1_B_M", "1_No"
string V2: "2_A_P", "2_A_M", "2_B_P", "2_B_M", "2_No"
string V3: "3_A_P", "3_A_M", "3_B_P", "3_B_M", "3_No"
[STRENGTHS]
default:2;
[SEEDS]
V1: "1_A_P", V2: "2_No", V3: "3_B_P";
[CONSTRAINTS]
V3=="3_B_M" -> V2!="2_B_M";
V3=="3_B_M" -> V1!="1_B_M";
V3=="3_B_P" -> V2!="2_B_P";
V3=="3_B_P" -> V1!="1_B_P";
```

The results generated by the combinatorial test case generation program and the PICT[16] execution results are shown in Fig. 4. There are 29 use cases generated by this method, and PICT generates 31 test cases. Furthermore, the syntax of this article is more concise, which avoids tools like PICT using complex syntax such as judgment statements to express parameter constraints.



- [8] Cohen, Myra B., et al. "Constructing Test Suites for Interaction Testing." International Conference on Software Engineering, 2003. Proceedings IEEE, 2003:38-48.
- [9] Cohen, D. M., et al. "The AETG System: An Approach to Testing Based on Combinatorial Design." IEEE Transactions on Software Engineering 23.7(1997):437-444.
- [10] Cohen, D. M., et al. "The Automatic Efficient Test Generator (AETG) system." IEEE (1994):303-309.
- [11] Cohen, David M., et al. "The Combinatorial Design Approach to Automatic Test Generation." Software IEEE 13.5(1996):83-88.
- [12] Tung, Yu Wen, and W. S. Aldiwan. "Automating test case generation for the new generation mission software system." Aerospace Conference Proceedings IEEE Xplore, 2000:431-437 vol.1.
- [13] Colbourn, Charles J., M. B. Cohen, and R. Turban. "A deterministic density algorithm for pairwise interaction coverage." Iasted International Conference on Software Engineering DBLP, 2004:345-352.
- [14] Bryce, René C, and C. J. Colbourn. "The density algorithm for pairwise interaction testing." Software Testing Verification & Reliability 17.3(2010):159-182.
- [15] Tai, K. C., and Y. Lie. A Test Generation Strategy for Pairwise Testing. IEEE Press, 2002.
- [16] Qin X, Duan J, Feng G, et al. Test Scenario Design for Intelligent Driving System Ensuring Coverage and Effectiveness[J]. International Journal of Automotive Technology, 2018, 19(4):751-758.

Specification-based Testing with Simulation Relations

Canh Minh Do, Kazuhiro Ogata

School of Information Science

Japan Advanced Institute of Science and Technology (JAIST)

1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

Email: {canhdominh,ogata}@jaist.ac.jp

Abstract—We propose a concurrent program testing technique that is a specification-based one and uses simulation relations from concurrent programs to formal specifications. For a formal specification S , a concurrent program P and a simulation relation r from P to S , the proposed technique is outlined as follows: (1) state sequences s_0, s_1, \dots, s_n are generated from P , (2) state sequences s'_0, s'_1, \dots, s'_m for S are obtained by converting s_0, s_1, \dots, s_n with r and (3) it is checked that S can accept s'_0, s'_1, \dots, s'_m . (1) is very crucial, but we first tackle (2) and (3) and then the present paper focuses on (2) and (3).

Keywords—concurrent program testing; Maude; meta-programming; simulation-based testing; simulation relations

I. INTRODUCTION

Major concepts of programming languages that can be used to write concurrent programs emerged in the 1980s and since nearly then studies on testing concurrent programs have been conducted. Arora, et al. have comprehensively surveyed testing concurrent programs [1]. They categorize it into eight classes: (a) reachability testing, (b) structural testing, (c) model-based testing, (d) mutation-based testing, (e) slicing-based testing, (f) formal method-based testing, (g) random testing, and (h) search-based testing. Model checking concurrent programs has been intensively studied, which may be classified into (c) and/or (f). Java Pathfinder (JPF) [2], [3] is such a model checker. Model checking is superior to the other testing techniques in that the former exhaustively checks all possible execution paths (or computations). However, model checking concurrent programs often encounters the notorious state explosion, which has not yet been conquered reasonably well.

We need a concurrent program testing technique that can scale reasonably well because most important software systems are in the form of concurrent programs, which are large-scale. We then propose a concurrent program testing technique in this paper toward this aim. The technique is a specification-based one. We suppose that programmers write concurrent programs based on formal specifications. The FeliCa team has demonstrated that use of formal specifications is useful as well as feasible in a practical setting [4].

This work was partially supported by JSPS KAKENHI Grant Number JP26240008 & JP19H04082.

DOI reference number: 10.18293/SEKE2019-027

Therefore, our assumption must be reasonable. Programmers need to comprehend formal specifications and must know their concurrent programs well and then they must be able to find some good relations between formal specifications and concurrent programs. Such relations should be simulation relations from the latter to the former. Then, we use such simulation relations to test concurrent programs.

Given a formal specification S , a concurrent program P and a simulation relation r from P to S , the proposed technique is outlined as follows: (1) state sequences s_0, s_1, \dots, s_n are generated from P , (2) state sequences s'_0, s'_1, \dots, s'_m for S are obtained by converting s_0, s_1, \dots, s_n with r and (3) it is checked that S can accept s'_0, s'_1, \dots, s'_m . (1) is very crucial, but we first tackle (2) and (3) and then the present paper focuses on (2) and (3).

Our approach uses formal specifications to test concurrent programs. Testing programs based on formal specifications has been studied [5]. Among such techniques are a CSP-based one [6] and an Event-B model-based on [7]. One difference between existing such techniques and our approach is that test cases are generated from formal specifications in the former, while the counterparts are generated from concurrent programs.

The rest of the paper is organized as follows: § II Preliminaries, § III Toward Concurrent Program Testing, § IV Specification Testing with Simulation Relations, § V Experiments, and § VI Conclusion.

II. PRELIMINARIES

A state machine $M \triangleq \langle S, I, T \rangle$ consists of a set S of states, the set $I \subseteq S$ of initial states and a binary relation $T \subseteq S \times S$ over states. $(s, s') \in T$ is called a state transition and may be written as $s \rightarrow_M s'$. Let \rightarrow_M^* be the reflexive and transitive closure of \rightarrow_M . The set $R_M \subseteq S$ of reachable states w.r.t. M is inductively defined as follows: (1) for each $s \in I$, $s \in R$ and (2) if $s \in R$ and $(s, s') \in T$, then $s' \in R$. A state predicate p is called invariant w.r.t. M iff $p(s)$ holds for all $s \in R_M$. A finite sequence $s_0, \dots, s_i, s_{i+1}, \dots, s_n$ of states is called a finite semi-computation of M if $s_0 \in I$ and $s_i \rightarrow_M^* s_{i+1}$ for each $i = 0, \dots, n-1$. If that is the case, it is called that M can accept $s_0, \dots, s_i, s_{i+1}, \dots, s_n$.

Given two state machines M_C and M_A , a relation r over R_C and R_A is called a simulation relation from M_C to

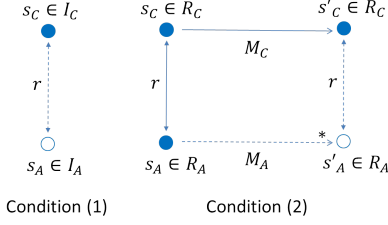


Figure 1. A simulation relation from M_C to M_A

M_A if r satisfies the following two conditions: (1) for each $s_C \in I_C$, there exists $s_A \in I_A$ such that $r(s_C, s_A)$ and (2) for each $s_C, s'_C \in R_C$ and $s_A \in R_A$ such that $r(s_C, s_A)$ and $s_C \rightarrow_{M_C} s'_C$, there exists $s'_A \in R_A$ such that $r(s_A, s'_A)$ and $s_A \rightarrow_{M_A}^* s'_A$ [8] (see Fig. 1). If that is the case, we may write that M_A simulates M_C with r . There is a theorem on simulation relations from M_C to M_A and invariants w.r.t M_C and M_A : for any state machines M_C and M_A such that there exists a simulation relation r from M_C to M_A , any state predicates p_C for M_C and p_A for M_A such that $p_A(s_A) \Rightarrow p_C(s_C)$ for any reachable states $s_A \in R_{M_A}$ and $s_C \in R_{M_C}$ with $r(s_C, s_A)$, if $p_A(s_A)$ holds for all $s_A \in R_{M_A}$, then $p_C(s_C)$ holds for all $s_C \in R_{M_C}$ [8]. The theorem makes it possible to verify that p_C is invariant w.r.t. M_C by proving that p_A is invariant w.r.t. M_A , M_A simulates M_C with r and $p_A(s_A)$ implies $p_C(s_C)$ for all $s_A \in R_{M_A}$ and $s_C \in R_{M_C}$ with $r(s_C, s_A)$.

States are expressed as braced soups of observable components, where soups are associative-commutative collections and observable components are name-value pairs in this paper. The state that consists of observable components oc_1 , oc_2 and oc_3 is expressed as $\{oc_1 \ oc_2 \ oc_3\}$, which equals $\{oc_3 \ oc_1 \ oc_2\}$ and some others because of associativity and commutativity. We use Maude [9], a rewriting logic-based computer language, as a specification language because Maude makes it possible to use associative-commutative collections.

Simple Communication Protocol (SCP), a communication protocol, is used as one running example in this paper. SCP consists of a sender, a receiver and two channels between them. One channel called dc (data channel) is a cell that is used to transfer pairs $\langle d, b \rangle$, where d is a data value and b is a Boolean value, to the receiver from the sender, and the other channel called ac (ack channel) is a cell that is used to deliver Boolean values (as ack) to the sender from the receiver. Both cells are unreliable in that the contents may drop. The sender maintains two pieces of information that are sb (sender bit) and data. sb is a Boolean value and data is the data to be delivered next to the receiver. The receiver maintains two pieces of information that are rb (receiver bit) and buf. rb is Boolean value and buf is the list of data received so far. Initially, sb is true, data is $d(0)$, rb is true,

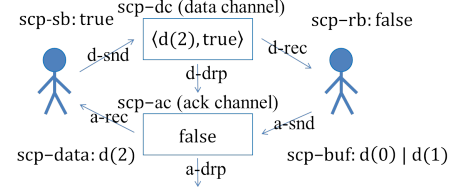


Figure 2. A state of SCP

buf is empty, dc is empty and cc is empty. The sender has two actions to do that are d-snd and d-rec. d-snd does the following: the pair $\langle \text{data}, \text{sb} \rangle$ is put into dc. d-rec does the following: if ac has a Boolean value b , then b is extracted and if $b \neq \text{sb}$, then data is set to the next data and sb is negated and otherwise nothing changes. The receiver has two actions to do that are a-snd and a-rec. a-snd does the following: rb is put into ac. a-rec does the following: if dc has $\langle d, b \rangle$, then $\langle d, b \rangle$ is extracted and if $b = \text{rb}$, then d is added to buf at the end and rb is negated and otherwise nothing changes. There are two more actions that are d-drp and a-drp. d-drp does the following: if dc is not empty, dc becomes empty. a-drp does the following: if ac is not empty, ac becomes empty. Fig. 2 shows a state of SCP.

A state of SCP is expressed as follows:

```
{(scp-sb: b1) (scp-data: d(n)) (scp-rb: b2)
 (scp-buf: dl) (scp-dc: cell1) (scp-ac: cell2)}
```

Each of the six actions in SCP is formalized as state transitions, which are described in Maude (conditional) rewrite rules (or rules) as follows:

```
rl [d-snd] : {(scp-sb: B) (scp-data: D)
 (scp-dc: DC) OCs}
=> {(scp-sb: B) (scp-data: D)
 (scp-dc: (< D, B >)) OCs} .
```

```
cr1 [a-rec1] : {(scp-sb: B) (scp-data: d(N))
 (scp-ac: B') OCs}
=> {(scp-sb: (not B)) (scp-data: d(N + 1))
 (scp-ac: empc) OCs} if B /= B' .
```

```
cr1 [a-rec2] : {(scp-sb: B) (scp-data: D)
 (scp-ac: B') OCs}
=> {(scp-sb: B) (scp-data: D) (scp-ac: empc)
 OCs} if B = B' .
```

```
rl [a-snd] : {(scp-rb: B) (scp-ac: AC) OCs}
=> {(scp-rb: B) (scp-ac: B) OCs} .
```

```
cr1 [d-rec1] : {(scp-rb: B) (scp-buf: Ds)
 (scp-dc: (< D, B' >)) OCs}
=> {(scp-rb: (not B)) (scp-buf: (Ds | D))
 (scp-dc: empc) OCs} if B = B' .
```

```
cr1 [d-rec2] : {(scp-rb: B) (scp-buf: Ds)
 (scp-dc: (< D, B' >)) OCs}
=> {(scp-rb: B) (scp-buf: Ds)
 (scp-dc: empc) OCs} if B /= B' .
```

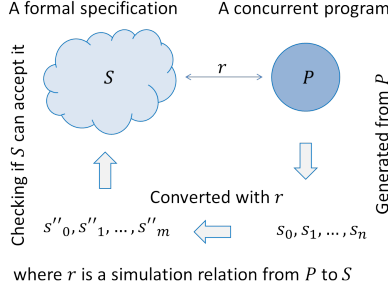


Figure 3. Specification-based concurrent program testing with a simulation relation

```

rl [d-drp] : {(scp-dc: P) OCs}
=> {(scp-dc: empc) OCs} .

rl [a-drp] : {(scp-ac: B) OCs}
=> {(scp-ac: empc) OCs} .

```

The search command given by Maude can conduct reachability analysis of state machines specified in Maude. Let s_1 and s_2 be the following states:

```

{(scp-sb: true)(scp-data: d(0))
(scp-rb: false)(scp-buf: d(0))
(scp-dc: < d(0), true >)(scp-ac: false)}

{(scp-sb: false)(scp-data: d(1))
(scp-rb: false)(scp-buf: d(0))
(scp-dc: < d(0), true >)(scp-ac: false)}

```

The search command “search [1,2] in SCP : $s_1 \Rightarrow^* s_2$.” checks if s_2 is reachable from s_1 in depth 2, where SCP is the module in which SCP is specified. If so, the command finds one transition sequence from s_1 to s_2 . Because there is such a transition sequence, the commands finds it. If [1, 1] is used instead of [1, 2], then because there is no transition sequence from s_1 to s_2 in depth 1, the command does not find any such transition sequences.

Maude has meta-programming (or reflexive programming) facilities with which we can develop software tools, such as Real-Time Maude. The first search command can be expressed in the term: `metaSearch(upModule('SCP, false), upTerm(s_1), upTerm(s_2), nil, '*', 2, 0)`, where `upModule` converts a module to its meta-representation, the term representing the module, and `upTerm` converts a term to its meta-representation. By reducing the term, we can essentially get the same result as the one obtained by the first search command.

III. TOWARD CONCURRENT PROGRAM TESTING

Testing concurrent programs is inherently different from testing sequential programs. All we need to test the latter is basically to check the output for each input, although there is some room to generate better test cases. Even though there is no decidable test oracle, we could use the metamorphic

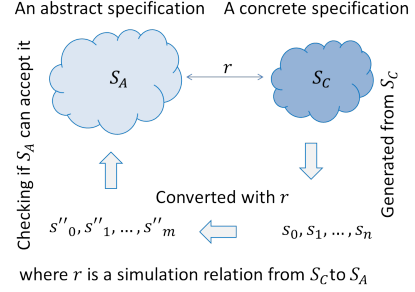


Figure 4. Specification-based specification testing with a simulation relation

testing technique. Concurrent programs have multiple active entities, such as threads, and therefore can lead to lots of execution scenarios. What execution scenario will be taken depends a scheduler, such as Java VM thread scheduler. It is usually impossible to control such a scheduler with an ordinal program. There may be some nondeterminism in concurrent programs. Hence, it does not suffice to have concurrent programs run on real machines to test such programs because we cannot check all possible execution scenarios.

One possible remedy is to use software model checkers, such as Java Pathfinder (JPF) [3]. JPF has its own VM running on a native Java VM and controls its own VM to cover all possible execution scenarios of concurrent programs. Many case studies with JPF have been reported and several techniques for JPF that may mitigate the notorious state explosion have been proposed. Even so, non-JPF experts can often encounter the state explosion that cannot be mitigated at all. We have written an ABP simulator in Java and tried to model check with JPF running on a computer that carries a 32GB memory the simulator in which each channel capacity is 3 and 3 messages are delivered to the receiver from the sender [10]. We have spent several days but the information we have obtained was out of memory. Thus, we do not think that it would suffice to use software model checkers, such as JPF, to test concurrent programs.

We propose a concurrent program testing technique that is a specification-based testing one (see Fig. 3). Let S be a formal specification of a state machine and P be a concurrent program. A state machine could be extracted from P , for example, as what is done by JPF. What we would like to do is to test if P is an implementation of S , or S simulates P . To this end, we use a simulation relation candidate r from P to S . For a formal specification S , a concurrent program P and a simulation relation r from P to S , the proposed technique does the following: (1) finite state sequences s_1, s_2, \dots, s_n are generate from P , (2) each s_i of P is converted to a state s'_i of S with r , (3) one of each two consecutive states s'_i and s'_{i+1} such that $s'_i = s'_{i+1}$ is deleted, (4) finite state sequences $s''_1, s''_2, \dots, s''_m$ are then obtained

and (5) it is checked that $s''_1, s''_2, \dots, s''_m$ can be accepted by S . We suppose that programmers write concurrent programs based on formal specifications, although it may be possible to generate concurrent programs (semi-)automatically from formal specifications in some cases. The FeliCa team has demonstrated that programmers can write programs based on formal specifications and moreover use of formal specifications can make programs high-quality. Therefore, our assumption is meaningful as well as feasible. If so, programmers must have profound enough understandings of both formal specifications and concurrent programs so that they can come up with simulation relation candidates from the latter to the former.

In our approach, state sequences generated from P are test cases. Therefore, it is really crucial that what state sequences are generated from P and how they are generated. The former and the latter have something to do with the quality of test cases and the scalability of our approach, respectively. Some may say that our approach has the same problems as software model checkers. Our approach never checks any properties while generating state sequences from P . It would take non-trivial time to check properties. Thus, we anticipate that our approach can scale better than software model checkers.

IV. SPECIFICATION TESTING WITH SIMULATION RELATIONS

This paper mainly focuses on the left part of the diagram shown in Fig.3. However, we need something, which is substituted for P , from which state sequences are generated. We use an abstract specification S_A as S and a concrete specification S_C as P in this paper (see Fig.4).

It is often the case that any state sequences generated from S_C cannot be accepted by S_A . A simulation relation r from S_C to S_A is not necessarily a function from S_C states to S_A states or from S_A states to S_C states in general. Because states in S_C are often designed by refining those in S_A , however, we conjecture that r is a function from S_C states to S_A states in practice.

Given a finite sequence s_0, s_1, \dots, s_n of states generated from S_C , each state is converted into a state in S_A with r , generating s'_0, s'_1, \dots, s'_n , where $s'_i = r(s_i)$ for each i and r is used as a function from S_C states to S_A states. There may be two consecutive states s'_i and s'_{i+1} such that $s'_i = s'_{i+1}$. If so, one of them is deleted. We then generate a sequence $s''_0, s''_1, \dots, s''_m$ of states in S_A such that there does not exist i such that $s''_i = s''_{i+1}$ (see the bottom part of Fig.4). We finally check if $s''_0, s''_1, \dots, s''_m$ is a finite semi-computation of S_A (see the left part of Fig.4).

Let sim be a function from S_C states to S_A . The function simList that converts s_0, s_1, \dots, s_n to $s''_0, s''_1, \dots, s''_m$ is defined as follows:

```
eq simList(C | L, S)
= if S == {empty}
```

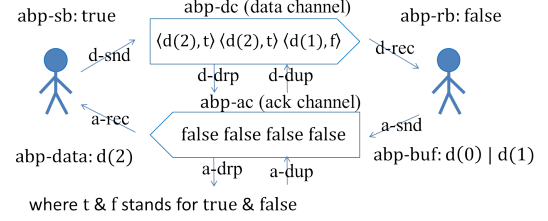


Figure 5. A state of ABP

```
then sim(C) | simList(L, sim(C))
else (
  if compareTo(sim(C), S)
  then simList(L, S)
  else sim(C) | simList(L, sim(C)) fi ) fi .
```

where $_|_$ is used as the constructor of state sequences. $\text{compareTo}(\text{sim}(C), S)$ checks if there are two consecutive states such that they are equal.

Given a module mQid in which a state machine is specified, two states $S1$ & $S2$ and the depth B , the function checkSttTrans checks if $S2$ is reachable from $S1$ in B w.r.t. the state machine, which is defined as follows:

```
ceq checkSttTrans(mQid, S1, S2, B)
= if sttTrans? :: ResultTriple
  then true else false fi
if sttTrans? :=
  metaSearch(upModule(mQid, false),
    upTerm(S1), upTerm(S2), nil, '*', B, 0) .
```

metaSearch is used to check if $S2$ is reachable from $S1$ in B w.r.t. the state machine specified as mQid .

Given a module mQid in which a state machine is specified, a sequence of the state machine states and a depth B , the function checkConform checks if the state sequence is a finite semi-computation of the state machine, which is defined as follows:

```
eq checkConform(mQid, S1 | S2 | L, B)
= $checkConform(mQid, S2 | L, S1, 0, B) .
eq $checkConform(mQid, nil, S, N, B)
= success .
eq $checkConform(mQid, S2 | L, S1, N, B)
= if checkSttTrans(mQid, S1, S2, B)
  then $checkConform(mQid, L, S2, N + 1, B)
  else {msg: "Failure", from: S1, to: S2,
    index: N, bound: B} fi .
```

$\text{checkSttTrans}(\text{mQid}, S1, S2, B)$ checks if $S1 \rightarrow^*_{\text{mQid}} S2$ in the depth B .

V. EXPERIMENTS

We use a specification of Alternating Bit Protocol (ABP) as S_C . ABP is a communication protocol as SCP. A state in ABP is shown in Fig.5. The difference between ABP and SCP is as follows: instead of the two cells used in SCP, ABP uses as two channels two queues that are unreliable in that

an element in the queues may drop and/or be duplicated. Therefore, there are two actions for each queue: d-drp and d-dup for dc and a-drp and a-dup for ac. There are totally eight actions in ABP.

A state of ABP is expressed as follows:

```
{ (abp-sb: b1) (abp-data: d(n)) (abp-rb: b2)
  (abp-buf: dl) (abp-dc: q1) (abp-ac: q2) }
```

Each of the eight actions in ABP is formalized as state transitions, which are described in Maude rules as follows:

```
rl [d-snd] : {(abp-sb: B) (abp-data: D)
  (abp-dc: Ps) OCs}
=> {(abp-sb: B) (abp-data: D)
  (abp-dc: (Ps | < D, B >)) OCs} .

crl [a-rec1] : {(abp-sb: B) (abp-data: d(N))
  (abp-ac: (B' | Bs)) OCs}
=> {(abp-sb: (not B)) (abp-data: d(N + 1))
  (abp-ac: Bs) OCs} if B /= B' .

crl [a-rec2] : {(abp-sb: B) (abp-data: D)
  (abp-ac: (B' | Bs)) OCs}
=> {(abp-sb: B) (abp-data: D)
  (abp-ac: Bs) OCs} if B = B' .

rl [a-snd] : {(abp-rb: B) (abp-ac: Bs) OCs}
=> {(abp-rb: B) (abp-ac: (Bs | B)) OCs} .

crl [d-rec1] : {(abp-rb: B) (abp-buf: Ds)
  (abp-dc: (< D, B' > | Ps)) OCs}
=> {(abp-rb: (not B)) (abp-buf: (Ds | D))
  (abp-dc: Ps) OCs} if B = B' .

crl [d-rec2] : {(abp-rb: B) (abp-buf: Ds)
  (abp-dc: (< D, B' > | Ps)) OCs}
=> {(abp-rb: B) (abp-buf: Ds) (abp-dc: Ps)
  OCs} if B /= B' .

rl [d-drp] : {(abp-dc: (Ps1 | P | Ps2)) OCs}
=> {(abp-dc: (Ps1 | Ps2)) OCs} .

rl [d-dup] : {(abp-dc: (Ps1 | P | Ps2)) OCs}
=> {(abp-dc: (Ps1 | P | P | Ps2)) OCs} .

rl [a-drp] : {(abp-ac: (Bs1 | B | Bs2)) OCs}
=> {(abp-ac: (Bs1 | Bs2)) OCs} .

rl [a-dup] : {(abp-ac: (Bs1 | B | Bs2)) OCs}
=> {(abp-ac: (Bs1 | B | B | Bs2)) OCs} .
```

Let `sim` be a simulation function that converts an ABP state

```
(abp-sb: S) (abp-data: D) (abp-rb: R)
  (abp-buf: BUFF) (abp-dc: DC2) (abp-ac: AC2)
```

to a SCP state

```
{(scp-sb: S) (scp-data: D) (scp-rb: R)
  (scp-buf: BUFF) (scp-dc: norm(hd(DC2)))
  (scp-ac: norm(hd(AC2))) }
```

where `hd` returns the top element if a given queue is not empty and an error element otherwise and `norm` returns the

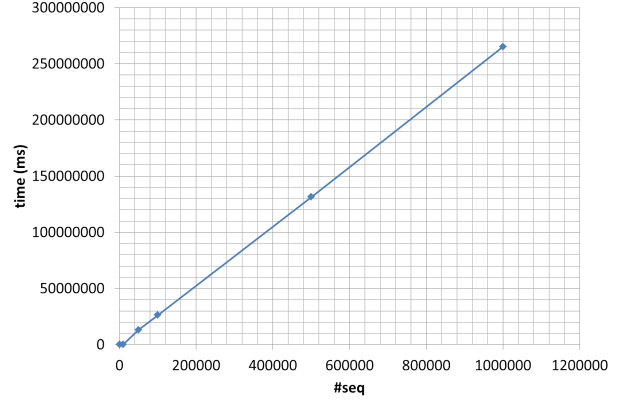


Figure 6. Time taken when the length of each state sequence is fixed (100) and the number of state sequences is changed (100, 1000, 10000, 50000, 100000, 500000 & 1000000)

cell in which the element is stored if the argument is an element and the empty cell if it is an error element.

Let `seqABP100` be a sequence of states randomly generated from the ABP specification such that its length is 100. Let `seqSCP` be the sequence of states obtained by `simList(seqABP100, {empty})`. We can check if `seqSCP` is a finite semi-computation of the SCP specification in the depth 2 by `checkConform('SCP', seqSCP, 2)`. The result is success. If we use 1 as the depth instead of 2, we get the following result:

```
{msg: "Failure", from: {scp-sb: true scp-data:
  d(0) scp-rb: false scp-buf: d(0) scp-dc:
  < d(0), true > scp-ac: false}, to: {scp-sb:
  false scp-data: d(1) scp-rb: false scp-buf:
  d(0) scp-dc: < d(0), true > scp-ac: false},
  index: 20, bound: 1}
```

This is because two state transitions need to be taken to move the state following `from:` to the state following `to:` in SCP as described in Sect. II.

It is also important to know how many state transitions need to be taken to move one state to the next state in the state sequence of S_A obtained by converting a state sequence of S_C with a simulation function. We also conjecture that programmers who have written a concurrent program based on a formal specification can guess such information because they need to understand the formal specification well and know a simulation relation from the concurrent program to the formal specification.

We used the SCP and ABP specifications in Maude to measure time taken to generate state sequences from the ABP specification, transform them with the simulation relation from ABP to SCP to other state sequences, and check if the state sequences obtained can be accepted by the SCP specification. We used one node of SGI UV3000

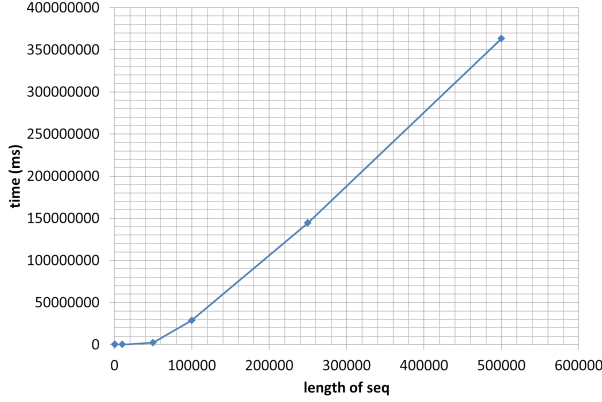


Figure 7. Time taken when the number of state sequences is fixed (1) and the length of the state sequence is changed (100, 1000, 10000, 50000, 100000, 250000 & 500000)

that carries 2.90GH microprocessor and 256GB memory for the experiments. Two sets of experiments were conducted. One set is to fix the length of each state sequence, which is 100, and modify the number of state sequences generated, which is one of 100, 1000, 10000, 50000, 100000, 500000 and 1000000. The other set is to fix the number of state sequences generated, which is one, and modify the length of the state sequence, which is one of 100, 1000, 10000, 50000, 100000, 250000 and 500000. For both sets of experiments, 2 was used as the depth of state transitions. Fig.6 shows the experimental results for the first set. The time taken increases almost linearly as the number of state sequences generated increases. Fig.7 shows the experimental results for the second set. The time taken increases a bit greater than linearly as the length of the state sequence generated increases.

VI. CONCLUSION

We have proposed a concurrent program testing technique that is a specification-based one and uses simulation relations from concurrent programs to formal specifications. For a formal specification S , a concurrent program P and a simulation relation from P to S , the proposed technique is outlined as follows: (1) state sequences s_0, s_1, \dots, s_n are generated from P , (2) state sequences $s''_0, s''_1, \dots, s''_m$ for S are obtained by converting s_0, s_1, \dots, s_n with r and (3) it is checked that S can accept $s''_0, s''_1, \dots, s''_m$. The present paper has focused on (2) and (3).

The first set of experiments (shown in Fig.6) indicates that it would be feasible to (almost) exhaustively check if state sequences whose length is small and that are generated from a concurrent program can be accepted by a formal specification with a simulation relation (candidate) from the program to the specification. This must be useful because of the small world hypothesis [11], which means that most flaws of programs lurk in a shallow depth and could be found

with such exhaustive testing in a shallow depth. The second set of experiments (shown in Fig.7) indicates that it would not be feasible to exhaustively test state sequences whose length is large and that are generated from a concurrent program with a formal specification and a simulation relation (candidate) from the program to the specification. There may be flaws lurking in programs in a non-shallow depth [12]. Therefore, it is worth testing state sequences whose length is large. It seems feasible to do so selectively. What and how long state sequences are selected is one piece of our future work.

The present paper does not mention anything about how to generate state sequences from concurrent programs. We plan to use Java as a programming language to write concurrent programs and to use JPF to generate state sequences from concurrent programs.

REFERENCES

- [1] V. Arora, R. K. Bhatia, and M. Singh, "A systematic review of approaches for testing concurrent programs," *Concurrency Computat.: Pract. Exper.*, vol. 28, no. 5, pp. 1572–1611, 2016.
- [2] K. Havelund and T. Pressburger, "Model checking JAVA programs using JAVA Pathfinder," *STTT*, vol. 2, no. 4, pp. 366–381, 2000.
- [3] W. Visser, K. Havelund, G. P. Brat, S. Park, and F. Lerda, "Model checking programs," *Autom. Softw. Eng.*, vol. 10, no. 2, pp. 203–232, 2003.
- [4] T. Kurita, M. Chiba, and Y. Nakatsugawa, "Application of a formal specification language in the development of the "Mobile FeliCa" IC chip firmware for embedding in mobile phone," in *FM 2008*, 2008, pp. 425–429.
- [5] M. Gaudel, "Software testing based on formal specification," in *PSSE 2007*, 2007, pp. 215–242.
- [6] A. Cavalcanti and M. Gaudel, "Testing for refinement in CSP," in *ICFEM 2007*, 2007, pp. 151–170.
- [7] D. H. Vu, A. H. Truong, Y. Chiba, and T. Aoki, "Automated testing reactive systems from Event-B model," in *4th NAFOS-TED Conf. Info. & Comp. Sci.*, 2017, pp. 207–212.
- [8] K. Ogata and K. Futatsugi, "Simulation-based verification for invariant properties in the OTS/CafeOBJ method," in *Refine 2007*, 2007, pp. 127–154.
- [9] M. Clavel, et al., *All About Maude*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4350.
- [10] K. Ogata, "Model checking designs with CafeOBJ – a contrast with a software model checker," Workshop on Formal Method and Internet of Mobile Things, ECNU, Shanghai, China, 2014.
- [11] D. Jackson, *Software Abstraction*. The MIT Press, 2012.
- [12] K. Ogata, M. Nakano, W. Kong, and K. Futatsugi, "Induction-guided falsification," in *8th ICFEM*, 2006, pp. 114–131.

A Survey Study on the Inference Problem in Distributed Environment

Adel Jebali

Tunis El Manar University,
Faculty of Mathematical Physical
and Natural Sciences of
Tunis, Tunisia
VPNC Laboratory
adel.jbali@fst.utm.tn

Salma Sassi

Jendouba University, Faculty of
Law Economics and
Management of Jendouba, Tunisia
VPNC Laboratory
salma.sassi@fsjegj.rnu.tn

Abderrazak JEMAI

Carthage University, Polytechnic
School of Tunisia,
SERCOM Laboratory,
INSAT, 1080, Tunis, Tunisia
abderrazekjemai@yahoo.co.uk

Abstract— Traditional access control models aim to prevent data leakage via direct accesses. A direct access occurs when a requester poses his query directly on the desired object. However, these models fail to protect sensitive data from being accessed with inference channels. An inference channel is produced by the combination of the legitimate response which a user receives from the system and metadata. Detecting and removing inference in database systems guarantee a high-quality design in terms of data secrecy and privacy. Parting from the fact that data distribution exacerbates inference problem, we give in this paper a survey of the current and emerging research on the inference problem in both centralized and distributed database systems and highlighting research directions in this field.

Keywords- Access Control, Inference Control, External Knowledge, Data Distribution, Secrecy and Privacy

I. INTRODUCTION

Access control models protect sensitive data from direct disclosure via direct accesses, however they fail to prevent indirect accesses [10]. Indirect accesses via inference channels occur when a malicious user combines the legitimate response that he received from the system with metadata. According to [11], external information to be combined with data in order to produce an inference channel could be database schema, system's semantics, statistical information, exceptions, error messages, user-defined functions and data dependencies. Detecting and removing inference in database systems guarantee a high-quality design in terms of data secrecy and privacy since this latter is considered as a new vision of the inference problem. Absolutely, this diversity of techniques to bypass access control mechanisms with inference channels has attracted considerable attention in recent years. A growing body of literature has examined the inference problem but no one of the proposed solutions seems to be the universal one. In reality, for each of the underlying techniques a specific solution has been proposed for handling each particular attack. There is consensus among security community that data distribution exacerbates inference problem. This is why several attempts have been done in the last two decades to

address this problem. This paper investigates current and emerging research on the inference control in centralized database systems, then it highlights inference in distributed environment. The reminder of this paper is organized as follows: Section 2 provides a brief description of research efforts on controlling inference in centralized database systems, section 3 review works on the inference control in distributed environment. Research directions are given in section 4. Finally, we conclude in section 5.

II. INFERENCE CONTROL IN CENTRALIZED DATABASE SYSTEMS

Traditional access control models aim to prevent data leakage via direct accesses. A direct access occurs when a requester poses his query directly on the desired object. However, these models fail to protect sensitive data from being accessed via indirect accesses [10]. An inference problem (also called inference aggregation problem) occurs when a user deduces sensitive information from a sequence of innocuous information in the database. It has been widely investigated in the literature since 1987 with the emergence of multilevel database systems. The first works in this field are presented in [16, 20, 22].

A. Inference Attacks and Prevention Methods

According to [10], there are three types of inference attack: Statistical attacks, semantic attacks and inference due to data mining. For each of the mentioned techniques, researchers have devoted a lot of efforts to deal with inference problem. For statistical attacks, techniques like Anonymization and Data-perturbation have been developed to protect data from indirect access. For security threats based on data mining, techniques like privacy-preserving data mining and Privacy-preserving data publishing was carried out. Furthermore, a lot of works have investigated the semantic attacks [3, 15, 20].

There exist in the literature more than one criteria to classify approaches that deal with inference. One proposed criteria is to classify these approaches according to data level and schema level [28]. In such classification, inference constraints are classified into schema constraints level and data constraints level. Other criteria could be according to

the time when the inference control techniques are performed. According to this criteria, the proposed approaches are classified in two categories: design time [7, 13, 14, 15, 20, 26] and query run time [1, 3, 11, 21, 22].

B. Discussion of the Inference Prevention Methods

The purpose of inference control at design time is to detect inference channels from earliest stage and eliminate them. These approaches provide a better performance for the system since no monitoring module is needed when the users query the database, by consequence improving query execution time. Nevertheless, design time approaches are too restrictive and may lead to over classification of the data. Besides, it requires that the designer has a good concept of how the system will be utilized. On the other hand, run time approaches provide data availability since they monitor the suspicious queries at run time. However, run time approaches lead to performance degradation of the database server since every query needs to be checked by the inference engine. Furthermore, the inference engine needs to manage a huge number of log files and users. As a result, this could induce slowing down query processing. In addition, run time approaches could induce a non deterministic access control behavior (users with the same privileges may not get the same response).

To summarise, the main evaluation criteria of these techniques is a trade-off between availability and system performance. We assert that the distribution of the data exacerbates the inference and privacy problems. In the next section we investigate the inference problem in distributed environment.

III. INFERENCE CONTROL IN DISTRIBUTED ENVIRONMENT

Inference control in distributed environment have been investigated from early 2000 until now. This field of study has received intention from researchers in database security, due to the fact that distribution aggravates inference problems and privacy concerns. In this section, we start by investigating research efforts on inference prevention in distributed database systems, then we review inference in data integration systems and discuss different works for mitigating this later. We survey inference problem in data integration systems through the *Mediator/Wrapper* architecture for the reason that this is the most suitable design to access distributed, heterogeneous and autonomous data sources. Additionally, we highlight inference prevention methods in data outsourcing scenario.

A. Inference Control in Distributed Database Systems

In [4], the authors have considered the inference problem where the data is combined from distributed database and released to the final users. In this situation of data dissemination, problem arises when non-sensitive attributes compromise sensitive attributes. According to presented work, one technique to mitigate inference is by modifying the non-sensitive data in the database.

Nevertheless, even with this modification, sensitive attributes still deducible when data from other databases is incorporated. The main idea behind this work is to not release certain non-sensitive information that can lead to probabilistic inference about the sensitive information while minimizing the loss of functionality. Consequently, the outputs are records that have been modified in order to anonymize sensitive attributes.

The authors of [24] have built on [4] to develop a work turning around inference prevention in distributed database systems. They proposed an inference prevention approach that enables each of the database in a distributed system to keep track of probabilistic dependencies with other databases and by consequence use that information to help preserve the confidentiality of sensitive data. The methodology is called "Agent-based" because every node in the distributed system is augmented with an agent to keep track of other nodes so that single point of failure and communication bottleneck are avoided. However, this approach has some limits. It treats the case where the distributed databases are overlapped (similar or have common attributes). Moreover, it assumes that the records in the distributed databases share the same keys constraints.

Inference problem have been also investigated in Peer-to-Peer environment through the work in [6]. The authors pinpoint the inference that occurs in homogeneous peer agent through distributed data mining and call this process peer-to-peer agent-based data mining systems. They assert that performing Distributed Data Mining (DDM) in such extremely open distributed systems exacerbates data privacy and security issues. As a matter of fact, inference occurs in DDM when one or more peer sites learn any confidential information about the dataset owned by other peers during a data mining session. The authors firstly classified inference attacks in DDM in two categories: inside attack scenario and outside attack scenario. After identifying DDM inference attacks, the authors propose an algorithm to control potential attacks (inside and outside attacks) to particular schema for homogeneous distributed clustering, known as *KDEC*. However, the algorithm proposed by the authors need to be improved from an accuracy point to expose further possible weakness of the *KDEC* schema.

B. Inference Control in Data Integration Systems

Inference control in data integration systems have been investigated in the last decade through the works in [12, 17, 18]. In such systems, a mediator is defined as a unique entry point to the distributed data sources. It provides to the user a unique view of the distributed data. From a security point of view, access control is a major challenge in this situation since the global policy must comply with the source policies. Complying with source policies means that a prohibited access at the source level should be also prohibited at the global level. [12, 17, 18] have demonstrated that despite the generation of a global policy at the mediator level that synthesizes and enforces the back-end data sources policies,

security breaches still possible via inference channel produced by semantic constraints. The problem is that the designer of the system cannot anticipate the inference channels that arise due to the dependencies that appear at the mediator level.

The first work attempting to control inference in data integration systems was introduced in [12]. The authors propose an incremental approach to prevent inference with functional dependencies. The proposed methodology includes three steps: synthesizing global policies, detection phase and Reconfiguration phase. In this work, authors have discussed only semantic constraints due to functional dependencies. Neither inclusion nor multivalued dependencies was investigated. Besides, other mapping approaches need to be discussed such as LAV and GLAV approaches.

The authors of [18] have inspired from [12] to propose an approach aiming to control inference in data integration systems. The proposed methodology resort to formal concept analysis as a formal framework to reason about authorization rules and functional dependencies as a source of inference. The authors adopt an access control model with authorization views and propose an incremental approach with three steps: generation of the global policy, global schema and global FD, Identifying disclosure transactions and Reconfiguration phase.

In [17] the authors have examined inference that arise in the web through RDF store. They propose a fine-grained framework for RDF data, then they exploit close graph to verify the consistency propriety of an access control policy when inference rules and authorization rules interact. Without accessing the data (at policy design-time), the authors propose an algorithm to verify if an information leakage will arise given a policy P and a set of inference rules R . Furthermore, the authors demonstrate the applicability of the access control model using a conflict resolution strategy (most specific takes precedence).

C. Inference and Data Outsourcing

Inference problem was not only investigated in previous distribution scenarios but also in data outsourcing. In this case, data owners place their data among cloud service providers in order to increase flexibility, optimize storage, enhance data manipulation and decrease processing time. Nonetheless, data security is widely recognized as a major barrier to cloud computing and other data outsourcing or Database-As-a-Service arrangements. Users are reluctant to place their sensitive data in the cloud due to concerns about data disclosure to potentially untrusted cloud providers and other malicious part [27]. It is from this perspective that inference problem was investigated in [2, 8].

In [2] authors resort to a controlled query evaluation strategy (CQE) to detect inference based on the knowledge of non-confidential information contained in the outsourced

fragments and priori knowledge that a malicious user might have. Regarding that CQE relies on logic-oriented view on database systems, the main idea of this approach is to model fragmentation logic-oriented too allowing for inference proofness to be proved formally even the semantic database constraints that an attacker may hold. Besides, vertical database fragmentation technique was considered by authors in [8] to ensure data confidentiality in presence of data dependencies among attributes. Those dependencies allow unauthorized users to deduce information about sensitive attributes. To tackle this issue, authors reformulate the problem graphically through an hypergraph representation and then compute the closure of a fragmentation by deducing all information derivable from its fragments via dependencies to identify indirect access. Nevertheless, the major limit of this approach is that it explores the problem only in single relational database.

IV. RESEARCH DIRECTIONS

Since the discussed works are recent, there are a number of concepts associated to security policies, privacy, data distribution and semantic constraints which could be considered to ensure better security and prevent inference from occurring in distributed environment. Hence, there are many research directions to pursue:

- Absence of modularity in data integration systems: in the case where a new source joins the system it is necessary to revise the global schema and the global policy. This is not suitable for distributed environment where the source joins and leaves the system continuously (e.g. Mobile environment).
- Authors deal only with semantic constraints represented by functional dependencies and probabilistic dependencies as a source of inference. However, other semantic constraints, example inclusion dependencies, join dependencies and multivalued dependencies should be considered as sources of inference.
- In data integration scenario, all approaches aim to handle inference at query run time by keeping track of the history of user queries and the current query. In the case where the system deals with a large volume of data and users number, run time approaches will lead to performance degradation by slowing down query processing, consequently, this may push the server (mediator) to bottleneck. Hence, design time approach should be adopted to overcome these problems since it is performed offline.
- Another weakness in these approaches is the negligence of collaborative inference. In fact, authors propose to block a sequence of violating transaction from being achieved to prevent the inference channel, but, what if this violating transactions results from a

combination of a set of queries from more than one user?

- Functional dependencies should be considered as a source of inference in data outsourcing scenario. although data dependencies may resemble functional dependencies, they model a different concept. In future work, we will present a study aiming to prevent inference from occurring in distributed cloud database. Our approach is graph-based that firstly detects inference channels caused by functional dependencies and secondly breaks those channels by exploiting vertical database fragmentation while minimizing dependencies loss.

V. CONCLUSION

This paper has surveyed the inference problem from two perspectives: centralized and distributed design. We first gave a review of current and emerging research about the inference control in centralized database systems, we have introduced different inference attacks and their prevention methods and discussed the trade-off between them. Furthermore, an insightful discussion about inference control in distributed environment was provided. We also pinpoint potential issues that are still unresolved. These issues are expected to be addressed in future work.

REFERENCES

- [1] Xiangdong An, Dawn Jutla, and Nick Cercone. 2006. Dynamic inference control in privacy preference enforcement. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*. ACM, 24.
- [2] Joachim Biskup, Marcel Preuß, and Lena Wiese. 2011. On the inference-proofness of database fragmentation satisfying confidentiality constraints. In *International Conference on Information Security*. Springer, 246–261.
- [3] Alexander Brodsky, Csilla Farkas, and Sushil Jajodia. 2000. Secure databases: Constraints, inference channels, and monitoring disclosures. *IEEE Transactions on Knowledge and Data Engineering* 12, 6 (2000), 900–919.
- [4] LiWu Chang and Ira Moskowitz. 2003. A study of inference problems in distributed databases. In *Research Directions in Data and Applications Security*. Springer, 191–204.
- [5] Yu Chen and Wesley W Chu. 2008. Protection of database security via collaborative inference detection. In *Intelligence and Security Informatics*. Springer, 275–303.
- [6] R Lopez de Mantaras and L Saina. 2004. Inference attacks in peer-to-peer homogeneous distributed data mining. In *ECAI 2004: 16th European Conference on Artificial Intelligence, August 22-27, 2004, Valencia, Spain: Including Prestigious Applicants [sic] of Intelligent Systems (PAIS 2004): Proceedings, Vol. 110*. IOS Press, 450.
- [7] Harry S. Delugach and Thomas H. Hinke. 1996. Wizard: A database inference analysis and detection system. *IEEE Transactions on Knowledge and Data Engineering* 8, 1 (1996), 56–66.
- [8] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Giovanni Livraga, Stefano Paraboschi, and Pierangela Samarati. 2014. Fragmentation in presence of data dependencies. *IEEE Transactions on Dependable and Secure Computing* 11, 6 (2014), 510–523.
- [9] Josep Domingo-Ferrer. 2002. Advances in inference control in statistical databases: An overview. In *Inference Control in Statistical Databases*. Springer, 1–7.
- [10] Csilla Farkas and Sushil Jajodia. 2002. The inference problem: a survey. *ACM SIGKDD Explorations Newsletter* 4, 2 (2002), 6–11.
- [11] Marco Guarnieri, Srdjan Marinovic, and David Basin. 2017. Securing Databases from Probabilistic Inference. In *Computer Security Foundations Symposium (CSF), 2017 IEEE 30th. IEEE*, 343–359.
- [12] Mehdi Haddad, Jovan Stevovic, Annamaria Chiasera, Yannis Velegarakis, and Mohand-Saïd Hacid. 2014. Access control for data integration in presence of data dependencies. In *International Conference on Database Systems for Advanced Applications*. Springer, 203–217.
- [13] Thomas H Hinke. 1988. Inference aggregation detection in database management systems. In *Security and Privacy, 1988. Proceedings., 1988 IEEE Symposium on. IEEE*, 96–106.
- [14] Thomas H Hinke, Harry S Delugach, and Randall P Wolf. 1997. Protecting databases from inference attacks. *Computers & Security* 16, 8 (1997), 687–708.
- [15] CE Landwehr and S Jajodia. 1992. The use of conceptual structures for handling the. inference problem. (1992).
- [16] Matthew Morgenstern. 1988. Controlling logical inference in multilevel database systems. In *Security and Privacy, 1988. Proceedings., 1988 IEEE Symposium on. IEEE*, 245–255.
- [17] Tarek Sayah, Emmanuel Coquery, Romuald Thion, and Mohand-Saïd Hacid. 2015. Inference leakage detection for authorization policies over RDF data. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 346–361.
- [18] Mokhtar Sellami, Mohand-Saïd Hacid, and Mohamed Mohsen Gammoudi. 2015. Inference control in data integration systems. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 285–302.
- [19] Jessica Staddon. 2003. Dynamic inference control. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. ACM, 94–100.
- [20] T-A Su and Gultekin Ozsoyoglu. 1991. Controlling FD and MVD inferences in multilevel relational database systems. *IEEE Transactions on Knowledge and Data Engineering* 3, 4 (1991), 474–485.
- [21] Bhavani Thuraisingham, William Ford, Marie Collins, and Jonathan O'Keeffe. 1993. Design and implementation of a database inference controller. *Data & knowledge engineering* 11, 3 (1993), 271–297.
- [22] MB Thuraisingham. 1987. Security checking in relational database management systems augmented with inference engines. *Computers & Security* 6, 6 (1987), 479–492.
- [23] Tyrone S Toland, Csilla Farkas, and Caroline M Eastman. 2010. The inference problem: Maintaining maximal availability in the presence of database updates. *Computers & Security* 29, 1 (2010), 88–103.
- [24] James Tracy, LiWu Chang, and Ira S Moskowitz. 2003. An agent-based approach to inference prevention in distributed database systems. *International Journal on Artificial Intelligence Tools* 12, 03 (2003), 297–313.
- [25] Hui Wang and Ruilin Liu. 2011. Privacy-preserving publishing microdata with full functional dependencies. *Data and Knowledge Engineering* 70, 3 (2011), 249 – 268.
- [26] Jingwen Wang, Jie Yang, Fen Guo, and Huaqing Min. 2017. Resist the Database Intrusion Caused by Functional Dependency. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2017 International Conference on. IEEE*, 54–57.
- [27] Xiaofeng Xu, Li Xiong, and Jinfei Liu. 2015. Database fragmentation with confidentiality constraints: A graph search approach. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. ACM, 263–270.
- [28] Raymond W Yip and EN Levitt. 1998. Data level inference detection in database systems. In *Computer Security Foundations Workshop, 1998. Proceedings. 11th IEEE. IEEE*, 179–189.

Towards human-centric software testing

Samantha Catania

Department of Computer Science
University of Malta
samantha.catania.12@um.edu.mt

Chris Porter

Department of Computer Information Systems
University of Malta
chris.porter@um.edu.mt

Mark Micallef

Department of Computer Science
University of Malta
mark.micallef@um.edu.mt

Abstract—Software testing is widely perceived to be the main activity in the software development process that provides confidence in the quality of a product prior to release. However, the term *software testing* itself provokes a multitude of different definitions and opinions as to the nature of the profession, the role of software testers and the utility of different processes and tools that come with the territory [1][2]. We argue that in order for researchers to effectively study the field and contribute to its progress, a consensus first needs to be reached about the entity being studied. In this paper we present an empirical study based on the modified Delphi card sort method involving four cohorts of testers in Malta and London. The result of this study is a consolidated consensus-based mental model outlining how software testers perceive their profession. This mental model can be used to align any future research efforts and tool development with testers’ own perception of their context.

Index Terms—Software testing, human factors and ergonomics, mental models

I. INTRODUCTION

“Ergonomics [or human factors] is the scientific discipline concerned with the understanding of interactions among humans and other elements of a system, and the profession that applies theory, principles, data and methods to design in order to optimise human well-being and overall system performance.” [3]

Testing is a human-intensive activity, requiring constantly changing information, in different formats, from various sources with different levels of access and over a variety of channels and tools. The level of experience and training background adds up to the complex nature of the domain, all of which contribute to high levels of cognitive workload. To date, and to the best of our knowledge, testing as a discipline in its various forms and shapes has never been treated scientifically from an ergonomics and human factors perspective. The terms ‘ergonomics’ and ‘human factors’ are interchangeable, however their use implies either the immediate physical environment (e.g. the workbench), or to the wider context in which work is carried out (e.g. the process), respectively [4].

In the past, we have attempted to treat this domain from both an ergonomics perspective (e.g. augmented reality in workbenches, just-in-time information and cognitive workload as well as from a human-factors perspective (e.g. the impact of training on tester performance [5], human-centred test

frameworks). Throughout this period we also immersed ourselves in practice within industry teams, as much as possible, to be able to experience and discuss issues with industry professionals.

It transpires that there is a more pressing need that needs to be tackled first before we are able to further this line of investigation. Due to the complexity of the domain, different people will have different and often wide-ranging perceptions and expectations of testing in practice and theory. Therefore, to be able to reason in terms of human factors and ergonomics in testing, we first need to build a clear picture of how testers reason about testing in general. This paper aims to achieve this. We therefore present a mental-model built following an empirical exercise with software testing professionals in Malta and London using the modified Delphi card sorting method. This mental-model aims at providing a common vocabulary through which we can study problems and propose solutions which would ultimately benefit software testers and their well-being.

A mental model or a conceptual model, represents the “cognitive shorthand” [6] of how a person, or a group of people, understand a complex product or domain. Looking at this from the perspective of a website, this conceptual model is enough to help users navigate and interact with the site, however if the implementation varies widely from the users’ mental model, users will find it close to impossible to reach their goals without having to think hard about their actions and how the site is reacting.

Although we understand that testing is complex in nature, we present a mental model based on a consensus-driven empirical exercise which elicits testers’ beliefs about the domain and its various aspects. This aims to inform human factors and ergonomics researchers and designers in their effort to contribute towards this domain. Through an understanding of how testing professionals perceive this domain, contributions could be better aligned and communicated to resonate with the nature of their work. A designer’s primary goal is to build user interfaces that map as closely as possible with the users’ mental models, abstracting away the internal complexities [7]. This same principle applies for research efforts, whereby we first need to understand the primary stakeholders’ beliefs and perceptions driving everyday work before being able to improve working conditions. Closing the gap between testers’ mental models and researchers’ perceptions or beliefs about the domain increases the chances of (a) improving the tester’s

well-being by solving the right problems at the right level, and (b) adoption of research outcomes in industry. As in design, research efforts need to be aligned with the user's mental model. For this to happen, researchers need to use their expertise in research and frame their efforts as closely to the testers' mental model as possible, building a model which bridges their knowledge and efforts with testers' perceptions. Cooper defines this as the "represented model" which is "one of the designer's most important goals". Cooper states that "it is critical that designers understand in detail how their target users think about the work they do with the software" [6].

II. SOFTWARE TESTING

Software testing is an activity predominantly associated with ensuring that a product meets a certain level of quality before it is released to customers. Despite its widespread use and numerous practitioners worldwide, software testing is arguably the least understood part of the development process [2]. In the two decades that we have studied and practised in the field, the lack of consensus on the nature of the field, its processes, tools and practitioners' role has been an ever palpable characteristic. Many conversations that we have had about software testing decisions in the industry have involved phrases like "*I do not believe in automated testing*", "*that is not what I think your role as a tester should be*" or "*we need to release tomorrow, can you test this quickly please?*".

Even definitions by respected authorities differ, albeit with some overlap. The British Standards Index refers to software testing as the activity of exercising software with the intent of finding errors and verifying that it satisfies specified requirements [8]. The IEEE has a similar definition but includes the notion that a system can be tested without being exercised (inspection). The ISTQB syllabus [9] (the de facto standard for tester certification) makes reference to testing being a measurement of software quality.

Within the field itself, there are a multiplicity of roles, taxonomies, strategies, techniques, processes and tools. The correct time, place and usage of each of these elements is the subject of frequent debate, which has also given rise to different schools of thought within the industry. Engineering-driven schools treat testing as a standards-driven industry which practitioners can consequently be trained and certified in. On the other end of the scale, the "Context-Driven" school of thought claims that the value of any practice depends on its context and the concept of a one-size-fits-all definition of the field and its practices is misguided [10].

In a widely cited paper charting the progress and future direction of the field, Bertolino [1] makes reference to the multiplicity of meanings that arise from the term "software testing", as well as the particular research challenges that this fact generates. Bertolino goes on to set out the field's achievements to date, ongoing challenges and future dreams. The first dream she outlines is that of the development of a unified theory of testing.

This lack of clarity is a contributing motivation behind this work. Before one can design research, tools and processes that

cater for the human tester more effectively, one first needs to understand the role and subsequent needs of the human tester. To this end, we undertook a consensus-based investigative approach as discussed in the following section.

III. METHODOLOGY

Motivated by the objective of establishing a consensus-based mental model amongst practitioners in the field about the nature of software testing, we selected a methodology centred around the Modified Delphi Card Sort method. The Delphi technique is a widely used and accepted method for gathering data from respondents within their domain of expertise [11]. Originally developed in the 1950s by Dalkey and Hemler [12], it has been widely used and adapted in various disciplines as a means of seeking out information that can generate consensus amongst participants.

Whereas initial versions of the technique involved using questionnaires with participants, we have chosen to use a card-sort variant whereby participants are asked to group concepts from their domain under categories using index cards. This can be done in one of two ways. The first is to utilise an open card-sort approach, whereby participants start with an empty slate and build a representation from there. The second way involves providing the first participant with a so-called *seeded deck*, that is, an initial model of the domain which she may choose to agree or disagree with in whole or in part. The participant can make any changes she sees fit by changing categories, adding terms and removing others. The result of the first participant's card sort are used as the seeded deck for the next participant, and so on. This technique has been shown to converge to a consensus with as little as 8-10 participants [13].

As depicted in Figure 1, two card sort exercises were carried out: one during a meeting of professional testers in Malta and another at a similar meeting organised by the British Computer Society in London.

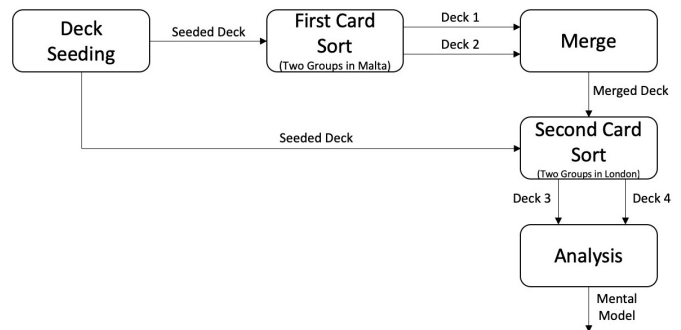


Fig. 1. An overview of the process used during the study.

Following an initial deck seeding exercise (Section III-A), two parallel card sort exercises were carried out (Section III-B) by two cohorts of participants, which resulted in two sorted decks that were subsequently merged (Section III-C). A second card sort (Section III-D) then took place, again with two parallel cohorts, one using the original seeded deck

and the other using the merged deck from the first card sort. This resulted in two final decks of cards which were fed into an analysis process (Section III-E) in order to produce the consolidated mental model. Each of these stages are discussed in turn below.

A. Deck Seeding

The seeded deck was compiled from a subset of terms found in a glossary published by the International Software Testing Qualifications Board (ISTQB) [14]. Given that the glossary contains over 600 terms, our initial processing of the glossary involved the removal of outlier terms and synonyms. Whilst outlier terms consisted of terms which we felt would not be helpful as part of the seeded deck (e.g. *Agile Manifesto* and *Myers-Briggs Type Indicator*), it is worth noting that should we be mistaken, it was entirely permissible for the terms to be reintroduced by participants during the card sort. At this point, we carried out a manual categorisation of the remaining terms into five broad categories: (1) *Testware* refers to entities that exist in the most part to enable and/or support the planning and execution of testing activities on a project; (2) *Artefact* refers to entities that are created, used and manipulated by testers or other stakeholders at some point during the testing process; (3) *Runtime*: refers to terms which represent entities that come into play in the period that a system is being executed as part of a test; (4) *Tools*: refers to devices, typically embodied as software systems that help support testers in a variety of tasks; and finally (5) *Test Strategy*, a category containing terms that identify different types of plans of action that testers can use to achieve their goal during software testing.

Once these categories were identified, we manually reduced the amounts of terms in each category based on our knowledge of the testing domain. Therefore, whereas a category called *Test Strategy* initially contained eighty terms, we manually reduced this to seven by keeping the most common strategies (e.g. *Acceptance Testing*) and removing the more obscure ones (e.g. *Monkey Testing*).

The resulting seeded deck consisted of five categories and twenty-five terms as shown in Table I.

TABLE I
THE CONTENTS OF THE INITIAL SEEDED DECK.

Category	Seeded Terms
Testware	Test Outcome, Test Data, Reports, Documentation, Test Suite, Test Script
Artefact	Features, Bugs, Code
Runtime	Configuration, System Under Test, Environment, Tester
Tools	Static Analysis Tools, Coverage Tools, Performance Testing Tools, Bug Tracking Tools, Automated Testing Tools
Test Strategy	Acceptance Testing, Ad Hoc Testing, Branch Testing, Exploratory Testing, Integration Testing, Performance Testing, Regression Testing

The materialisation of this seeded deck enabled us to move on to the next stage of the study.

B. First Card Sort - Malta

Armed with two copies of the seeded deck, we attended a gathering of software testing professionals in Malta and recruited sixteen participants, who we split evenly into two cohorts. Each participant was asked to reflect on their mental model of software testing and subsequently spend as much time as needed to modify the deck so that it accurately fit his/her mental model. This was done by any combination of (1) *adding* new terms or groups; (2) *moving* cards between groups; and (3) *removing* terms or groups which they considered not to be a fit for purpose. When a participant was finished, the next participant was asked to come in and continue the exercise.

At the end of the exercise, the two decks were analysed and merged into a single deck as discussed in the following section.

C. Merging of Decks

The two decks resulting from the parallel card sorts carried out in Malta were merged using a five-step process as follows. The first step involved *discarding replicated terms* such that terms which were added multiple times in the same deck were earmarked for discarding so that only one instance of each term remained. In cases where the terms appeared in different categories, reference to the ISTQB Glossary [14] was made in an effort to choose the category that best fit the emerging mental model. The next step involved *renaming of synonyms* to make sure that any instances whereby testers were referring to the same concept using different labels, one term was selected and used in place of all the synonyms. We then turned our attention to categories and automatically *retained any categories that were common in both decks* for the merged deck. Similarly we also *retained terms that appeared in both decks*. If the terms appeared in different groups, a judgement was made as to which group was the best fit for that term. Finally, we *discarded any empty categories*. Any categories that contained no terms following the preceding steps in the merge were discarded.

The two card sort exercises in Malta both retained the original five categories from the seeded deck and added four groups between them. However, the merging process resulted in the four new categories having no terms and were consequently removed. This resulted a merged deck with the characteristics described in Table II.

TABLE II
THE NUMBER OF INITIAL (I), RETAINED (R), ADDED (A), DELETED (D) AND FINAL (F) TERMS FOLLOWING THE FIRST CARD SORT AND MERGE.

Category	I	R	A	D	F
Testware	6	5 (83%)	2 (33%)	1 (17%)	7 (+17%)
Artefact	3	2 (67%)	5 (166%)	1 (33%)	7 (+133%)
Runtime	4	3 (75%)	2 (50%)	1 (25%)	5 (+25%)
Tools	5	4 (80%)	8 (160%)	1 (20%)	12 (+140%)
Test Strategy	7	5 (71%)	7 (100%)	2 (29%)	12 (+71%)
Totals:	25	19 (76%)	24 (96%)	6 (24%)	43 (+72%)

The figures in Table II indicate that there was an overall increase of 72% in the size of the mental model with about

24% of our initial seeded deck being rejected by participants. The highest relative increases were in the *Tools* category (+140%) and the *Artefact* category (+133%).

D. Second Card Sort - London

In order to further refine the emerging mental model, two cohorts of seven volunteers attending a one-day software testing conference organised by the British Computer Society in London were tasked with carrying out a further card sort. One cohort started with the original seeded deck whilst the other one started with the merged deck. The exact same protocol that was used in the first card sort was utilised in the second card sort with the results being fed into the final stage of the study.

E. Analysis

The card sort exercises generated 83 terms in total and each was either seeded, added, removed or moved at one or more points during this study. In order to make sense of this data, a scoring system was devised whereby every time participants interacted with a term, the term would gain or lose a certain amount of points. Therefore, when a term was seeded by a researcher, added by a participant or removed by a participant, it gained 1 point, 2 points and lost one point respectively. Whilst these values may seem arbitrary, they were designed to (1) characterise each term's journey through the study into a single score; and (2) assign more importance to terms which were added by participants without any prior mention.

The result was a ranked list of 83 terms in five categories which was then analysed by researchers with a view of establishing a cutoff point in each category that would indicate which terms should be included in the consolidated mental model. The cutoff decision was made based on (1) identifying points in the list where a significant scoring gap occurred between one term and the next; and (2) manually removing terms which objectively did not fit the category or were synonyms for other terms that were already included. The result was a mental model with the same categories as those designed in the original seeded deck.

IV. DISCUSSION

The mental model resulting from the research is represented in the form of a mind map in Figure 2. In this section, we discuss characteristics of this mental model and comment on its convergence, evolution and its implications on testers' perception of their field.

A. Convergence

Consistent with other studies that used the Modified Delphi Card Sort method, participants appear to converge towards a consensus relatively quickly. As shown in Table III, whereas the first card sort resulted in an increase in deck size from 25 to 43 terms (72%), there was only a net decrease of 2 terms (-5%) in deck size following the second card sort. Also, as shown in Table IV, more terms were retained, and less were added or removed as research progressed from the first to the second card sort.

TABLE III
COMPARING MENTAL MODEL SIZE FROM THE SEEDED DECK (Δ_s) THROUGH TO THE MERGED DECK (Δ_m) AND THE FINAL MENTAL MODEL.

Category	Seeded #	Merged #	Δ_s	Final #	Δ_s	Δ_m
Testware	6	7	+17%	5	-17%	-29%
Artefact	3	7	+133%	10	+233%	+43%
Runtime	4	5	+25%	6	+50%	+20%
Tools	5	12	+140%	9	+80%	-25%
Test Strategy	7	12	+71%	11	+57%	-8%
Totals:	25	43	+72%	41	+64%	-5%

TABLE IV
TERMS RETAINED, ADDED AND DELETED BETWEEN SUCCESSIVE DECKS.

	Seeded \rightarrow Merged	Merged \rightarrow Final
Retained	19 (76%)	37 (86%)
Added	24 (+96%)	4 (+9%)
Deleted	6 (-24%)	6 (-14%)
Net Churn:	18 (+72%)	-2 (-5%)

B. How the seeded model evolved

The contents of the Testware category remained mostly unchanged with no terms being added and one term (*Documentation*) being moved to the Artefacts category. The Artefacts category itself grew threefold in size from three terms to nine with practitioners strongly backing terms that the original seeded model had left out. Interestingly, the term *Features* was consistently removed by participants but was replaced by multiple other terms such as *Specification* and *Acceptance Criteria*.

The Runtime category retained all but one of its seeded terms (*Tester*) and doubled in size from three to six terms with the testers introducing the terms *Mocking*, *User* and *Bugs*.

Finally, practitioners retained most of the terms from the Tools and Test Strategies categories, removing two terms from each category but also supplemented each category considerably. Six new types of tools were added, as well as six new testing strategies. All terms added received consistent backing from participants in successive card sorts.

C. Interesting Observations

The process of obtaining information about the mental model of software testers from practitioners themselves provided some interesting insights.

Firstly, testers do not see themselves as part of their own mental model. The term *Tester* was presented to three cohorts as a result of being part of the seeded deck. However, it was removed every single time. This indicates to us that contrary to our view that the tester should be at the centre of any discussion related to the testing process, the testers take themselves out of the equation. This flies contrary to many discussions we witnessed in the industry whereby testers would express frustration at not being included enough in the software development process.

Secondly, testers need to somehow effectively use as many as eleven different testing strategies in their day-to-day job. This is complemented by nine different types of testing tools,

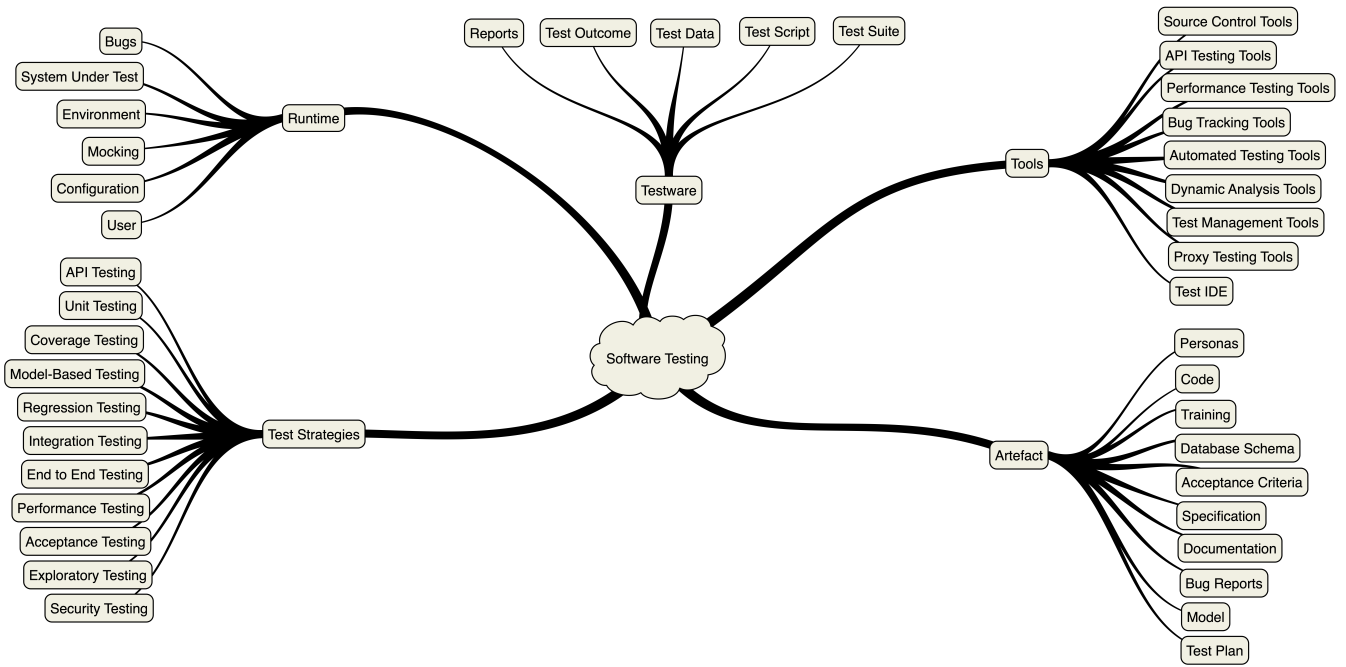


Fig. 2. The tester’s mental model that resulted from the card sorts.

which hopefully make the testers’ lives easier. This indicative of the highly complex nature of the testing profession.

In the multiple opportunities that presented themselves throughout this study (from seeding through multiple card sorts), the term *Usability Testing* was only mentioned once and did not make it into the consolidated mental model. This is interesting because it indicates that testers do not perceive usability testing as a core part of their job. Rather, they are concerned with ensuring that the product meets the stated specifications, even if the specifications do not necessarily provide the customer with the experience they desire. Given the ease with which customers can move to competitors in today’s dynamic online markets, this type of thinking could be counterproductive and end up producing high quality software that customers do not want [15].

V. RELATED WORK

In this section, we compare and contrast our work with similar research efforts. Different works in the literature concerned with understanding the software testing domain exist for one of three reasons: (1) understanding the application of testing in a specific domain; (2) knowledge management; or (3) forming a more coherent understanding for pedagogical purposes.

With regards to domain-specific work, Nasser et al. [16] present an ontology based on state machine based testing whilst Sapna and Mohanty’s [17] work focuses on scenario-based testing. Yu et al. [18] focus on understanding software testing as a service (TAAS). Whilst such studies have value at forming an in-depth understanding of testing within individual domains, our work is more focused on forming a wider practitioner-oriented understanding of the field. Nevertheless,

it is interesting to note how the organisation of different models differs based on their focus. For example, Yu et al. [18] proposed categories such as *Test Type*, *Target Under Test*, *Test Environment* and *Test Schedule*. Whilst these categories bare resemblance to those in our work and the work of others, the emphasis can be seen to focus on scenarios whereby testing is perceived (and even sold) as an outsourced service.

Focusing on increasing testcase reuse through knowledge management, Guo et al. [19] develop an ontology centred solely around the *test case*. They develop and propose the use of their unified standard format for test cases and argue that this can promote reuse. Focusing on web services, Bai et al. [20] propose a so-called Test Ontology Model that models testing artefacts and relationships between them. Barbosa et al. [21] propose *OntoTest* a collection of six sub-ontologies of testing named as *Testing Process*, *Testing Phase*, *Testing Artifact*, *Testing Step*, *Testing Resource*, and *Testing Procedure*.

Perhaps the work that is most closely related to this paper is that by Arnicans and Straujums [22] who propose a hierarchical model of the testing domain constructed from the 800 entries in the ISTQB Glossary [14], the same source used for creating the seeded deck in our study. Using a technique whereby each term in the glossary was assigned a weight and subsequently related to other words, they converted the ISTQB Glossary into a browsable hierarchical concept map. They found that the ‘weightiest’ nine terms were *testing*, *test*, *tool*, *software*, *process*, *analysis*, *capability*, *technique* and *coverage*. These top-level terms contain 425 (70%) of the glossary’s entries between them. There are some similarities between Arnicans and Straujums’ top terms and the categories in our mental model but the mapping is not direct. For example, the

term *tool* maps to our *tools* category whilst *software* maps to *artefact* and *technique* maps to *test strategies*. However, the terms contained in each of these mapped categories are not the same. One example is that some techniques that we refer to as strategies, are referred to as being part of the *process* category in Arnicans and Straujum's work. One should note that the scope behind the work differs from ours in that whilst Arniscans and Straujum are concerned with making the ISTQB Glossary more understandable, we are interested in understanding the mental model held by practitioners in the field.

VI. CONCLUSIONS AND FUTURE WORK

We motivated this study by arguing that since testing is a human-intensive activity, the human tester needs to be at the centre of research contributions to the field. This implies that testing, as a discipline, would benefit from being treated scientifically from an ergonomics and human-factors perspective. However, our initial work in this area uncovered a lack of clarity as to what exactly constitutes software testing from the tester's perspective.

As a result of the study presented in this paper, we propose a mental model elicited from practitioners using a consensus building approach. Having this mental model available provides researchers with an insight into how testing practitioners perceive their professional context and can thus form as a basis for aligning research efforts with practitioners' views. The

A. Future Work

With regards to future work, we would like to pursue two main paths of research. Firstly, we would like to continue to validate the model through further card sorts in order to reduce external threats to validity whilst also gaining a deeper understanding about whether cohorts of testers with specific characteristics (e.g. experience testers or testers working in a specific domain) would diverge from the model. Our initial discussions with peers about the results presented here resulted in questions regarding issues such as how the background of participants might have affected the results or why certain terms do not appear. It would be interesting to investigate these questions and also repeat these exercises on a regular basis to understand if and how practitioner perceptions evolve. Secondly, we would like to revisit our work on treating software testing from an ergonomics and human factors perspective (e.g. augmented reality workbenches, just-in-time information and cognitive workload) in light of the mental model and its implications. This will help us refocus such efforts so as to increase the chances of (a) improving the tester's well-being by solving the right problems at the right level, and (b) adoption of research outcomes by industry stakeholders.

mental model by no means covers the whole testing domain but we argue that aligning research efforts to this model is more likely to provide value to the practitioners who created it.

REFERENCES

- [1] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 85–103.
- [2] J. A. Whittaker, "What is software testing? and why is it so hard?" *IEEE software*, vol. 17, no. 1, pp. 70–79, 2000.
- [3] I. E. Association, "Definition and domains of ergonomics," 2016. [Online]. Available: <https://www.iea.cc/whats/>
- [4] C. I. of Ergonomics and H. Factors, "What is ergonomics? find out how it makes life better." [Online]. Available: <https://bit.ly/2SoLsJz>
- [5] M. Micallef, C. Porter, and A. Borg, "Do exploratory testers need formal training? an investigation using hci techniques," in *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2016, pp. 305–314.
- [6] A. Cooper, R. Reimann, and D. Cronin, *About face 3: the essentials of interaction design*. John Wiley & Sons, 2007.
- [7] J. Nielsen, "Mental models," 10 2010. [Online]. Available: <https://www.nngroup.com/articles/mental-models>
- [8] S. C. Reid, "Bs 7925-2: The software component testing standard," in *apaqs*. BSI, 2000, p. 139.
- [9] A. Roman, "2018 foundation syllabus overview," in *A Study Guide to the ISTQB® Foundation Level 2018 Syllabus*. Springer, 2018, pp. 3–11.
- [10] C. Kaner and J. Bach, "What is context-driven testing," 2009.
- [11] C.-C. Hsu and B. A. Sandford, "The delphi technique: making sense of consensus," *Practical assessment, research & evaluation*, vol. 12, no. 10, pp. 1–8, 2007.
- [12] N. Dalkey and O. Helmer, "An experimental application of the delphi method to the use of experts," *Management science*, vol. 9, no. 3, pp. 458–467, 1963.
- [13] A. Soranzo and D. Cooksey, "Testing taxonomies: beyond card sorting," *Bulletin of the Association for Information Science and Technology*, vol. 41, no. 5, pp. 34–39, 2015.
- [14] I. ISTQB, "Glossary of testing terms," *ISTQB Glossary* <http://www.istqb.org/downloads/finish/20/193.html>, 2015.
- [15] E. Ries, *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Crown Books, 2011.
- [16] V. H. Nasser, W. Du, and D. MacIsaac, "Knowledge-based software test generation," in *SEKE*, 2009, pp. 312–317.
- [17] P. Sapna and H. Mohanty, "An ontology based approach for test scenario management," in *International Conference on Information Intelligence, Systems, Technology and Management*. Springer, 2011, pp. 91–100.
- [18] L. Yu, L. Zhang, H. Xiang, Y. Su, W. Zhao, and J. Zhu, "A framework of testing as a service," in *2009 International Conference on Management and Service Science*. IEEE, 2009, pp. 1–4.
- [19] S. Guo, J. Zhang, W. Tong, and Z. Liu, "An application of ontology to test case reuse," in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*. IEEE, 2011, pp. 775–778.
- [20] X. Bai, S. Lee, W.-T. Tsai, and Y. Chen, "Ontology-based test modeling and partition testing of web services," in *2008 IEEE International Conference on Web Services*. IEEE, 2008, pp. 465–472.
- [21] E. F. Barbosa, E. Y. Nakagawa, and J. C. Maldonado, "Towards the establishment of an ontology of software testing," in *SEKE*, 2006, pp. 522–525.
- [22] G. Arnicans and U. Straujums, "Transformation of the software testing glossary into a browsable concept map," in *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering*. Springer, 2015, pp. 349–356.

Semantic Analysis for Deep Q-Network in Android GUI Testing

Tuyet Vuong*, Shingo Takada*

*Dept. of Information and Computer Science, Keio University, Yokohama, Japan

{tuyet, michigan}@doi.ics.keio.ac.jp

Abstract—Since the big boom of smartphone and consequently of mobile applications, developers nowadays have many tools to help them create applications easier and faster. However, efficient automated testing tools are still missing, especially for GUI testing. We propose an automated GUI testing tool for Android applications using Deep Q-Network and semantic analysis of the GUI. We identify the semantic meanings of GUI elements and use them as an input to a neural network, which through training, approximates the behavioral model of the application under test. The neural network is trained using the Q-Learning algorithm of Reinforcement Learning. It guides the testing tool to explore more often functionalities that can only be accessed through a specific sequence of actions. The tool does not require access to the source code of the application under test. It obtains higher code coverage and is better at fault detection in comparison to state-of-the-art testing tools.

Keywords: *Automated Android Testing; GUI Testing; Reinforcement Learning, Deep Q-network*

I. INTRODUCTION

In 2017, the number of monthly active Android devices reached a new milestone of 2 billion [1], accompanied by about 3 million applications in the Google Play Store [2]. The mobile application market is on-demand and so competitive that a small bug in the software can cause users to uninstall the product and opt for another. Assuring application quality is hence very important. Even though a lot of effort has been made to develop automated testing techniques, we still heavily rely on manual testing in practice, with only 3% of developers fully engaging in automation [3]. Developers also admitted in recent surveys that Graphical User Interface (GUI) testing is especially labor-intensive, time-consuming, and challenging to automate because GUI tests must be constantly rewritten when changes, even small, are made [3].

Android application's GUIs contain various types of *components* such as button, text input, slide bar, switch, etc. Moreover, users can interact with each type of component in multiple ways through *events*: click, long click, swipe, scroll, etc. This complexity makes GUI testing techniques like random testing inefficient because it attributes a uniform probability distribution to all combinations of components and events, while an efficient testing strategy should select and follow the specific paths that reveal application's functionalities.

In our previous work [4], we presented an automated testing tool implementing the classic Q-Learning algorithm of reinforcement learning, which demonstrated positive improvements in code coverage. In this paper, we continue to follow the reinforcement learning approach and propose two main contributions:

- Improve the learning of the application's behavioral model by analyzing the semantic representations of GUI components.
- Use Deep Q-Network [5] to approximate the behavioral model of the application under test.

The remainder of this paper is organized as follows: Section II reviews related work in Android automated testing and the application of reinforcement learning in software testing. Section III provides a brief introduction to Deep Q-Network. Section IV elaborates our proposed approach and implementation details. Section V analyzes the evaluation results and finally section VI concludes the paper along with suggestions for future works.

II. RELATED WORK

In recent years, researchers have tried different approaches to test Android applications, among the most popular trends are random testing, model-based testing, and heuristic-based testing. Tools such as Puma [6] and A3E [7] build a model of the application under test then systematically explore the application based on this model. On the other hand, Evodroid [8] and Acteve [9] use special algorithms such as symbolic execution and evolutionary algorithms to test the application. As our paper proposes a black-box GUI testing method, we investigate further in black-box GUI testing tools.

Monkey [10] is a random event generator which is embedded in the Android Development Toolkit. It is therefore commonly used thanks to its simplicity and availability. It performs tests by sending thousands of events per second to the application and usually obtains high code coverage in comparison to other testing tools [11]. Despite the high code coverage, faults discovered by Monkey are hard to locate because tests are hard to reproduce and closer to stress tests than functionality tests. Dynodroid [12] improves random testing strategy by analyzing the context of the application then executes the most relevant event at each step (RandomBiasedStrategy). Developers can provide inputs such as authentication information beforehand to the tool to

unblock some steps. Dynodroid is also capable of generating system events by analyzing the listeners of the application.

Reinforcement Learning was used in software testing in the past and has shown its ability to improve random exploration strategy. Mariani et al. proposed a tool called AutoBlackTest [13], a black-box GUI testing tool for Java desktop software. The tool uses Q-Learning algorithm to build a behavioral model of the software, represented as a multi-directional graph. Using this behavioral model, the tool can plan its exploration route in order to get to hard-to-reach GUIs (which can only be reached through a specific sequence of actions). TESTAR [14] also uses Q-Learning to generate test sequences based on GUI. The Q-Learning algorithm was proven to be beneficial, provided that we choose an adequate set of parameters.

There are currently two main approaches when applying reinforcement learning to Android testing. The first approach is proposed by Koroglu, et al. [15], where they train a single matrix across multiple apps using random exploration then use it to test other applications. The Q-value distribution matrix is trained for two objectives: increasing activity coverage and crash detection. Overall, their tool obtained higher activity coverage and number of distinct crashes compared to other state-of-the-art testing tools. The second approach is presented in our previous work [4] and the work of D. Adamo et. al [16], where a unique set of Q-value is calculated for each application.

III. DEEP Q-NETWORK

Reinforcement Learning (RL) is a field in Artificial Intelligence where an agent learns to behave optimally in its environment through trial-and-error interactions [17], step by step. At each time step t , it observes the state s_t of the environment and takes an action a_t based on its policy π . The environment then transitions to a new state s_{t+1} based on s_t and a_t . It also outputs a scalar reward r_{t+1} as feedback that the agent then uses to update its knowledge. The goal of the agent is to learn a policy π^* that maximizes the expected cumulative reward of a sequence of actions in the environment.

The reinforcement learning problem can be formulated as a Markov decision process [17], defined by:

- A set of possible states: S
- A set of possible actions: A
- A reward function for the next state given a (state, action) pair: $R(s_t, a_t, s_{t+1})$
- A transition probability i.e distribution over the next state given a (state, action) pair: $T(s_{t+1}|s_t, a_t)$
- A discount factor $\gamma \in [0, 1]$, where lower γ emphasizes more on immediate rewards.

There are two major directions in solving RL problems: algorithms based on value functions and algorithms based on policy search [18]. Q-Learning falls into the first category. For each policy π , we define an action-value function or quality function (Q-function). The value $Q^\pi(s_t, a_t)$ is the expected cumulative reward that can be achieved by executing a sequence of actions that starts with an action a_t

from a state s_t and then follows the policy π . The optimal Q-function Q^* is the maximum expected cumulative reward achievable for a given (state, action) pair, over all possible policies.

$$Q^*(s_t, a_t) = \max_{\pi} \sum_{t \geq 0} (\gamma^t r_t | s = s_t, a = a_t, \pi) \quad (1)$$

Intuitively, if the optimal quality function Q^* is known, at each step s_t , the optimal strategy is to take the action that maximizes the sum: $r + \gamma Q^*(s_{t+1}, a_{t+1})$ where r is the immediate reward of the current step. This is known as the Bellman equation in dynamic programming [19]:

$$Q^*(s_t, a_t) = R(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \quad (2)$$

The Q-learning algorithm uses equation (2) to estimate the value of Q^* iteratively. The Q-function is initialized with a default value. Every time the agent executes an action a_t from state s_t to reach state s_{t+1} and receives a reward r_{t+1} , the Q-function is updated as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (3)$$

In this formula, $\alpha \in [0, 1]$ is the learning rate and regulates the impact of a new observation on the estimated values. The Q-learning algorithm is guaranteed to converge to Q^* if applied to a Markovian environment, with a bounded immediate reward and with state-action pairs continually updated [20].

The main motivation for the birth of Deep Reinforcement Learning (DRL) is to scale the classic RL problems to more complex state and action spaces [5] [18]. In brief, DRL trains a neural network to approximate the optimal policy and/or optimal value functions. In the case of Q-Learning, the state-value function Q is estimated by a Deep Q-Network (DQN) with weight w :

$$Q(s_t, a_t, w) \approx Q^\pi(s_t, a_t) \quad (4)$$

The weight of the DQN is updated based on a loss function defined as:

$$L(w) = (r_t + \gamma \max_a Q(s_{t+1}, a, w) - Q(s_t, a_t, w))^2 \quad (5)$$

which leads to the following Q-Learning gradient:

$$(r_t + \gamma \max_a Q(s_{t+1}, a, w) - Q(s_t, a_t, w)) \frac{\partial Q(s_t, a_t, w)}{\partial w} \quad (6)$$

The stochastic gradient descent method can then be used to minimize the loss $L(w)$ and the Q-Network will gradually converge toward the optimal Q-function Q^* . In the next section, we explain how we use Deep Q-Network to incorporate the semantic representations of GUI states to the reinforcement learning algorithm.

IV. PROPOSED APPROACH: SEMANTIC ANALYSIS OF GUI AS AN INPUT FOR DEEP Q-NETWORK IN ANDROID GUI TESTING.

In our previous work [4] we used Q-Learning algorithm as an exploration strategy to test Android applications. We

dynamically built a behavioral model of the application under test while interacting with it. With classic Q-Learning, we calculated and constantly updated a dictionary holding the Q-value of each pair of (state, event). Even though the experiment demonstrated positive results, we noticed two main weaknesses in our model:

- The purpose of the reinforcement learning algorithm is to guide the exploration toward revealing and testing the application's hard-to-reach functionalities. However, a path that reveals the application's functionalities should consider the semantic meaning of the components upon which we take action. Take the example of an alarm clock application: the sequence of actions that reveals the functionality *Change timezone* should be described as *click on a menu component - scroll down the list - click on setting - click on change timezone button*, instead of simply *click - scroll - click - click*. The latter sequence of events would have very different outcomes when being executed on different sets of components.
- The algorithm cannot scale well when the number of states, GUI components in each state, and events increase. Mobile applications are becoming more and more complex and the Android OS can now support more and more gestures (events). Our testing tool should be able to handle large and complex GUIs.

The first point is addressed by a semantic analysis of GUI components that we elaborate in the following section. As for the second point, we propose using Deep Q-network to potentially solve the complexity problem of reinforcement learning.

A. Semantic classification of Android GUI components

GUI testing tools usually interact with the application under test by sending events to UI components but few of them have considered the meaning of the components they interact with. QBE [15] took some initiatives by separating the actions on hard buttons of the phone from the on-screen events (click, long-click, etc). However, they didn't consider any semantic meaning of the GUI components on the screen. In our approach, we use the semantic information of GUI components as an input to guide the exploration.

Recent works of T.F. Liu et. al presented a guideline to identify the semantic meaning of mobile app GUI [21]. They established a lexical database of 25 types of UI components, 197 text button concepts and 135 icon classes. We're interested in the classification of UI components in particular, where components are separated into groups such as Input, List Item, Toolbar, Background Image, etc. They employed a code-based heuristics approach: using a lexical database, they classified a component by examining its Android class and the classes of its parent components.

In our testing tool, we use the same method to classify components. However, instead of classifying all visible components, we only classify actionable components, which are either clickable, long-clickable, scrollable or checkable. If a parent component is actionable, then all of its children nodes are also actionable of the same type. We reduce the number

TABLE I
CLASSIFICATION OF UI COMPONENTS

Group	Class name
Input	EditText, SearchBox, AutoCompleteTextView, AutoSuggestView, Field, Input, CheckBox, DatePicker, RadioButton, CheckedTextView, Switch, SeekBar
Navigation	ToolBar, TitleBar, ActionBar, Menu, Navigation, SideBar, Drawer, AppBar, TabWidget
List	ListItem, ListView, RecyclerView, ListPopUpWindow, GridView, GroupView
Button	Button, GlyphView, TextView, ImageView

of component types down to four main groups: Navigation, Input, Button, and List, as these group can cover the majority of actionable components [21]. The reference for classifying each group is presented in Table I, inspired by the one provided by T.F. Liu et. al.

We identify UI components in Navigation and List group by their parent nodes. In other words, once we identify a List or Navigation component by its class name, all of its children fall under the same group respectively. The Input group should be understood in a large sense, including all components that receive information from the user. Finally, only actionable TextView and actionable ImageView are classified as Button (Text Button and Image Button respectively).

B. Deep Q-Network as an Android Testing Tool

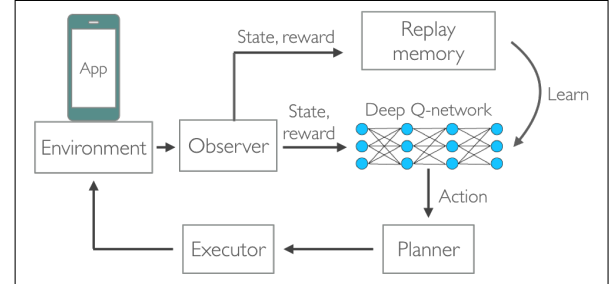


Fig. 1. The overall architecture

Our testing tool QDroid, whose architecture is presented in Figure 1, consists of 5 main modules: Environment, Observer, Deep Q-Network (including a replay memory), Planner, and Executor. QDroid interacts with the application under test step by step in order to estimate a behavioral model of the application, represented by the Deep Q-network. The goal of the testing tool is to generate test cases that test application's hard-to-reach states, cover the most code possible, and reveal faults, all in a limited amount of time.

A test case is created when the testing tool executes a sequence of actions in the application under test, this sequence is also known as an *episode* in reinforcement learning. In our implementation, an episode ends when one of the following conditions is met:

- It reaches the maximum number of 20 steps (transitions). After an empirical study where we vary the episode's length from 10 to 50, the value 20 was chosen because it gave the best average coverage across apps.
- Its last action leads to exiting the application.

- The screen is frozen (no changes in GUI) for the last 10 steps.

After an episode, the environment is reset, and the testing tool jumps to a random activity of the application. We reset the environment by killing all running processes in the emulator, then uninstalling and reinstalling the application. Each step of an episode proceeds as follows:

- 1) The *Observer* observes the application under test and builds the abstract representation of the GUI (the current *state*).
- 2) The *Observer* converts the current state into an input that we then use to feed to the Deep Q-Network.
- 3) The *Deep Q-Network* outputs a probability distribution over the next component that we should act on and passes it to the *Planner*.
- 4) The *Planner* chooses the next component to execute based on an $\epsilon - greedy$ policy.
- 5) The *Executor* executes an event on the chosen component.
- 6) The *Environment* transitions to a new state and returns the reward of the transition.
- 7) The transition, consisting of the old state, the new state, the executed component, and the reward, is added to the *Replay Memory*.
- 8) The *Deep Q-network* updates its weight by learning from a sampled batch of transitions from the *Replay Memory*.

This workflow is similar to the one in our previous work because both of them follow the Q-Learning algorithm. Nevertheless, the main difference with this architecture is concerned with the Deep Q-Network and the Observer, which also leads to changes in the Executor and the Planner. The implementation details of each module are given below:

1) *Environment*: The Environment is an interface that allows us to interact with the Android emulator and the application under test. Besides communicating with the Observer and the Executor, its most important role is to hold a function that calculates the reward value for each transition. We consider that a transition is better than another if it triggers more UI changes [13] [4]. Given two states s_1 and s_2 , the reward function calculates the degree of change from s_1 to s_2 by counting the number of GUI events in s_2 but not in s_1 , described as $|s_2 \setminus s_1|$. The relative change is then defined by the ratio $|s_2 \setminus s_1| / |s_2|$ where $|s_2|$ is the number of GUI events in $|s_2|$.

2) *Observer*: During run time, the Observer extracts the current screen's GUI hierarchy using Android UI Automator [22], obtaining the GUI tree. It then analyzes the GUI tree and classifies all actionable UI components into semantic categories as presented in Table I. It creates an abstract representation of the screen, also known as a *state*. A state consists of the activity's name and a set of GUI components. Each component is represented by a tuple containing the following information: semantic group, coordinates, class name, resource id, and four booleans indicating if the component is clickable, long-clickable, scrollable, checkable. The Observer

only acknowledges components that belong strictly to the application under test in order to assure the accuracy of the model. However, it also takes into account two hard buttons of the phone: the menu button and the back button, which are important and necessary to navigate in the application. The two hard buttons are classified in the Navigation semantic group.

The Observer creates a vector of length 4 where the elements hold the number of components belonging to the semantic groups given in Table I. The vector is then passed as an input of the DQN.

3) *Deep Q-Network and replay memory*: Following Mnih, et. al's guidelines [8], we integrate a Deep Q-Network into our testing tool while simplifying its structure, because the dimension of our input is much smaller than in the case of learning from raw images. Our Deep Q-Network has two fully connected layers, each one is followed by ReLu activation and the loss function is defined as in section III. We implement a Replay Memory with a capacity of 500 transitions and a target Q-network which is updated every 20 learning steps. We train the Deep Q-Network every 5 transitions with a batch of 32 transitions sampled randomly from the Replay Memory.

4) *Planner*: The Planner is the principal actor that guides the testing tool to explore the application. It receives a probability distribution over the four GUI component groups and decides which component to act on next based on a $\epsilon - greedy$ policy. The policy selects a random component with the probability ϵ and follows the prediction of the Deep Q-Network with the probability $1 - \epsilon$. At the beginning of the testing process, we want to encourage exploration behavior so that we can rapidly populate the state space. After a certain number of episodes when the Deep Q-network has collected enough transitions and has gained a certain knowledge about the application under test, we encourage the exploitation behavior: using the knowledge of the DQN to navigate in a more intelligent way. Hence, we decided to set $\epsilon = 1$ at the first episode and gradually decrease ϵ to 0.5 over the first 100 episodes (equivalent to about 2000 transitions). The value of ϵ is then maintained constant until the end of the testing process. When the Planner acts according to the Deep Q-Network prediction, it selects a component on the screen that has the highest probability value. For example, if the DQN outputs the probability distribution for (Input, Navigation, List, Button) respectively as (0.6, 0.2, 0.1, 0.1), the Planner will look for a component of group Input first. If there is not any component of group Input, it will look for a component of group Navigation and continue in the same manner until finding a component.

5) *Executor*: Unlike the executor in the testing tool of our previous work, which knows exactly which event to send to which component on screen, in this approach the executor only receives the information of which component on the screen that it needs to take action on. Based on the component's semantic group and clickable/long-clickable/checkable/scrollable information, the executor decides the type of event that it needs to send to the component.

For example, if the component is in the group Input and class EditText, the executor generates a random text to fill the input field. If the component is in the List group and scrollable, the executor executes a scrolling event. The testing tool currently supports 7 types of events: click, long click, scroll up, scroll down, swipe left, swipe right, text input.

V. EVALUATION

A. Overview

We aim to answer three research questions:

- RQ1: Does QDroid achieve higher code coverage than our previous approach and state-of-the-art testing tools?
- RQ2: Does QDroid achieve high code coverage faster than other state-of-the-art testing tools?
- RQ3: Can the tool reveal faults during test? Is it better than other tools ?

We measure code coverage (line and method coverage) and count the number of distinct faults as metrics for evaluation. We compare the results of QDroid with our previous work (ClassicQ) [4] and state-of-the-art testing tools: Dynodroid (Dyno) [12], Puma [6], A3E [7] and GuiRipper (GuiR) [23].

We use AndroTest [11], a framework for comparing different automated testing tools to set up virtual machines and run each testing tool separately. Each virtual machine runs Ubuntu 32-bit, has 6114Mb of base memory and 2 processors. Each testing session is run for 2 hours, on a fresh Android emulator with all the data from the previous session removed. AndroTest also provides a set of instrumented open-source applications. Because a lot of apps provided are either too simple (containing only one activity with few GUI changes) or fail to start before the testing begins, we selected 12 apps which are stable for most of the testing tools.

After each test session on an application, QDroid provides developers with:

- The record of each action taken during each episode, which can be used to reconstruct test cases.
- The evolution of coverage during test.
- Android execution log, which is used to detect faults.
- The number of crashes occurred during the test.

B. Results

1) *RQ1: Does QDroid achieve higher code coverage than our previous approach and state-of-the-art testing tools?*: Table II gives details of the average method coverage obtained by each testing tool on each application. It also calculates the p-value of the hypothesis that QDroid obtains higher code coverage than each of the other testing tools in average. Fig. 2 shows the distribution of line and method coverage across target applications for QDroid, ClassicQ, Dynodroid, Puma, A3E and GuiRipper.

QDroid performs better than our previous tool ClassicQ, but it was not statistically significant ($p = 0.14$). As for other tools (Dynodroid, Puma, A3E and GuiRipper), QDroid obtained higher code coverage at a statistically significant level ($p < 0.05$). Its best and worst coverage is also higher than the best and worst coverage of all the other four tools.

TABLE II
METHOD COVERAGE AFTER TWO HOURS (%) AND P-VALUE FOR QDROID PERFORMS STATISTICALLY SIGNIFICANT BETTER THAN EACH OF THE STATE-OF-THE-ART TESTING TOOLS

Application	Qdroid	ClassicQ	Dyno	Puma	A3E	GuiR
Any Memo	35.09	35.36	15.39	-	-	3.63
My Expenses	64.18	42.50	29.56	35.24	15.05	17.04
Who has my stuffs	89.01	88.09	58.93	67.48	51.15	29.92
Tippy Tipper	56.11	55.75	46.61	52.49	52.49	20.54
Munch Life	53.85	60.00	83.08	61.54	50.00	43.85
Mini Note Viewer	54.55	40.21	20.85	35.25	5.02	7.18
Mileage	35.29	34.73	26.85	34.06	2.99	16.38
Multi SMS sender	43.71	36.13	41.33	33.06	15.32	31.29
Hot Death	64.34	13.13	29.80	39.10	3.66	37.24
Random Music Player	58.73	58.73	58.73	58.73	3.17	41.27
Dalvik Explorer	83.51	83.40	23.82	70.87	41.08	5.39
Weight Chart	38.35	18.64	35.73	16.50	-	7.28
Mean	56.39	47.22	39.22	42.03	19.99	21.75
Standard deviation	16.64	22.02	18.80	20.23	20.98	13.96
p-value	-	0.14	0.02	0.047	0.0008	0.00009

One exception is with Munch Life where QDroid performs worse. In this case, we need to repeat the same action several times on the screen where the GUI skeleton doesn't change to be able to unlock a new state. Because QDroid favors the actions that trigger changes in UI and the test case is designed to end early when no GUI change is detected, QDroid isn't able to unlock the new state.

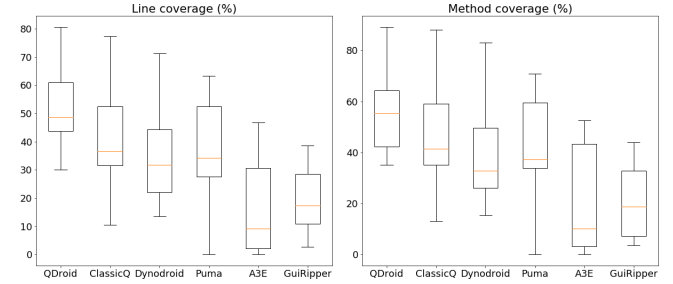


Fig. 2. Distribution of average coverage of each testing tool across apps

2) *RQ2: Does QDroid achieve high code coverage faster than state-of-the-art testing tools?*: The evolution of code coverage for QDroid, Dynodroid, Puma, A3E and GuiRipper (the average across all target applications and all runs) is presented in Fig. 3. On average, QDroid kick-starts with a high coverage and its performance improves over time. This proves that QDroid learns the behavioral model of the application gradually, and exploits this model more and more at the later phase of the testing process.

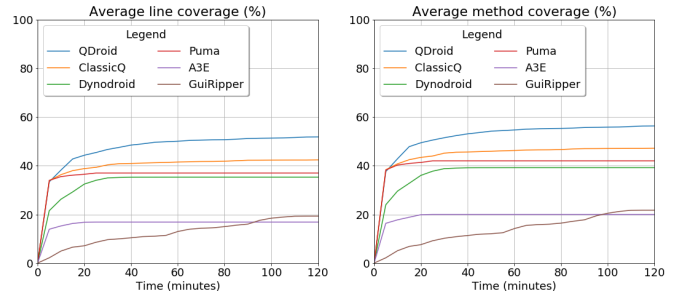


Fig. 3. Evolution of code coverage during test

TABLE III
NUMBER OF UNIQUE FAULTS DISCOVERED ACROSS ALL RUNS

App	Qdroid	Dyno	Puma	A3E	GuiR
Any Memo	11	9	0	0	0
My Expenses	0	0	0	0	0
Who has my stuffs	0	0	0	0	0
Tippy Tipper	0	0	0	0	0
Munch Life	0	0	0	0	0
Mini Note Viewer	1	1	1	0	0
Mileage	1	0	0	0	0
Multi SMS sender	0	0	0	0	0
Hot Death	2	0	0	0	0
Random Music Player	0	0	0	0	0
Dalvik Explorer	1	0	0	0	0
Weight Chart	1	1	0	0	0

3) *RQ3: Fault detection ability*: Table III gives the number of distinct faults discovered by each testing tool. The number of faults in each application before test is unknown. Distinct faults are identified by their unique error messages in the emulator's log. QDroid is able to detect faults during tests and outperforms the four other tools. The majority of the exceptions discovered belong to six main classes *java.lang*, *java.io*, *java.net*, *android.database*, *android.content* and *android.system*. Notice that even though different testing tools can have the same number of faults for one application, the faults are not necessarily identical. This is the case of Mini Note Viewer where the faults discovered by QDroid, Dynodroid and Puma are all different. For Any Memo and Weight Chart, the faults discovered by QDroid and Dynodroid have overlaps. Note that the framework we used, AndroTest, did not specifically state how many faults existed in each of the apps. Thus, we cannot determine the fault detection percentage of each tool. Nevertheless, QDroid performed the same or better compared to all other tools.

C. Threats to validity

The number of applications used in our evaluation is limited, hence raises a threat to external validity. Even though the target applications are chosen of different sizes and categories, it is unsure that the result can be generalized.

Training of a deep neural network can give very different outcome between different runs, which introduces a threat to internal validity. We ran each testing tool five times and took the average as the final result to reduce this threat.

VI. CONCLUSION

In this paper, we present a new approach using semantic analysis of GUI components and Deep Q-Network to conduct tests on Android applications. Our automated testing tool QDroid analyses the semantic information of GUI components and uses it as input to train a Deep Q-Network. The neural network, based on the principle of reinforcement learning, estimates a behavioral model of the application and exploits this model to guide the exploration inside the application. Evaluation has shown that QDroid presents an improvement in code coverage in comparison to state-of-the-art testing tools and it is effective in detecting faults.

As the integration of Deep Q-Network in the testing tool shows positive result, future work on this topic includes an

evaluation of QDroid on applications having more complex GUI. Specifically, we plan on extending the action space (by combining event and component as one action) and the state space (for example, adding more information to distinguish similar states).

REFERENCES

- [1] Google's announcement: <https://twitter.com/Google/status/864890655906070529>.
- [2] Total Apps on Google Play: <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>.
- [3] M. E. Joorabchi, A. Mesbah and P. Kruchten, "Real Challenges in Mobile App Development". Proc. of ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, 2013, pp. 15-24.
- [4] Thi Anh Tuyet Vuong and Shingo Takada. 2018. "A reinforcement learning based approach to automated testing of Android applications". Proc. of A-TEST 2018, 31-37.
- [5] Mnih et al. "Human-level control through deep reinforcement learning" Nature. 518. 529-33, 2005.
- [6] Shuai Hao, Bin Liu, Suman Nath, William G.J. Halfond, and Ramesh Govindan. 2014. "PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps". Proc. of MobiSys 2014, 204-217.
- [7] Tanzirul Azim and Iulian Neamtii. 2013. "A3E - Targeted and Depth-first Exploration for Systematic Testing of Android Apps". Proc. of OOPSLA 2013, 641-660.
- [8] R. Mahmood, N. Mirzaei, and S. Malek. 2014. "EvoDroid: segmented evolutionary testing of Android apps" Proc. of FSE 2014, 599-609.
- [9] S. Anand, M. Naik, M. J. Harrold, and H. Yang. 2012. "Automated concolic testing of smartphone apps". Proc. of FSE 2012, Article no. 59.
- [10] Android Monkey: <https://developer.android.com/studio/test/monkey.html>
- [11] S. R. Choudhary, A. Gorla and A. Orso, "Automated Test Input Generation for Android: Are We There Yet?", Proc. of ASE 2015, pp. 429-440.
- [12] A. Machiry, R. Tahiliani, and M. Naik. 2013. "Dynodroid: an input generation system for Android apps", Proc. of ESEC/FSE 2013, 224-234.
- [13] L. Mariani, M. Pezze, O. Riganelli and M. Santoro, "AutoBlackTest: Automatic Black-Box Testing of Interactive Applications", Proc. of 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012, pp. 81-90.
- [14] Anna I. Esparcia-Alcazar, Francisco Almenar, Urko Rueda Mirella Martinez, and Tanja E.J. Vos. 2016. "Q-learning strategies for action selection in the TESTAR automated testing tool". Proc. of META 2016. 174-180
- [15] Y. Koroglu et al., "QBE: QLearning-Based Exploration of Android Applications", Proc. of ICST 2018, pp. 105-115.
- [16] David Adamo, Md Khorrom Khan, Sreedevi Koppula, and Renée Bryce. 2018. "Reinforcement learning for Android GUI testing". Proc. of A-TEST 2018, 2-8.
- [17] Richard S. Sutton and Andrew G. Barto (Eds.). 1998. "Reinforcement Learning An Introduction". MIT Press, Cambridge, MA.
- [18] Arulkumaran, Kailash, Marc Peter Deisenroth, Miles Brundage and Anil A. Bharath. "Deep Reinforcement Learning: A Brief Survey." IEEE Signal Processing Magazine 34 (2017): 26-38.
- [19] Richard Bellman. "On the Theory of Dynamic Programming". PNAS, 38(8): 716-719, 1952.
- [20] Christopher J.C.H. Watkins and Peter Dayan. 1992. "Technical Note: Q-Learning". Machine Learning 8, 3-4 (May 1992), 279-292.
- [21] Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018. "Learning Design Semantics for Mobile Apps" Proc. of UIST 2018, 569-579.
- [22] UI Automator for Python: <https://github.com/xiaocong/uiautomator>
- [23] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon. 2012. Using GUI Ripping for Automated Testing of Android Applications. Proc. of ASE 2012, 258-261.

Impacts of Data Uniformity in the Reuse of Acceptance Test Glue Code

Douglas Hiura Longo, Patrícia Vilain and Lucas Pereira da Silva

Informatics and Statistics Department,
Federal University of Santa Catarina,
Florianopolis, Brazil

douglas.hiura@posgrad.ufsc.br, patricia.vilain@ufsc.br, lucas.pereira@ufsc.br

Abstract— This paper presents the design and results of an experiment to evaluate the impact/effect of data uniformity in automation of acceptance tests. An experiment to specify acceptance tests, represented by the User Scenarios through User Interaction Diagrams (US-UIDs) format, with non-technical user has been set up involving two projects. In the first project, called P1, the treatment of data uniformity is held by an expert, while in the second project, called P2, no treatment of data uniformity is done. In both projects, automation of acceptance tests was developed for evaluation and comparison of the following artifacts: data uniformity, fixture name sharing, automation time, and glue code volume. The results show that there is a meaningful statistics difference of uniformity between projects P1 and P2, where P1 resulted in a better uniformity. However, although the treatment of data uniformity does not show meaningful statistics difference according to the strategy of fixture names sharing used, the time spent in fixture naming was more than two times higher in P2. In addition, the glue code volume was less than half in P1 comparing to P2.

Keywords-component: *Acceptance Test; Agile Software Development; Glue Code; Uniformity; Reuse; US-UID.*

I. INTRODUCTION

In the agile software development, acceptance tests are adopted to enable the communication and collaboration among the stakeholders [1, 2, 3]. Some agile developers, users and clients use acceptance testing as a way of specifying software requirements instead of using only common artifacts based on natural language [17, 18]. This is an attempt to improve the quality of requirements because several problems can arise when requirements are written in natural language, for example, readers and writers can use the same word for different concepts, or express the same concept in completely different ways [18]. In addition, it is estimated that 85% of software defects are originated from ambiguous, incomplete or illusory software requirements [19].

Acceptance tests can be specified using semi-structured formats like user stories (such as Behavior Driven Development - BDD) [4, 5, 6], tables (such as Fit tables [3, 7]), or diagrams (such as US-UIDs [8, 9, 10]). However, the development process of automated acceptance testing is more complex than just specifying acceptance tests in a chosen format. Independent of the acceptance test format, it is also necessary to implement the

code that binds each acceptance test to the System Under Test (SUT). This code is known as glue code [16].

In general, acceptance tests are designed to exercise different functions of the SUT. However, in order to put the SUT in a suitable state for execution, the glue code of different acceptance tests may call some common SUT functions. In this way, the reuse of the glue code is important to reduce the time and cost of the maintenance. In line with Borg and Kropp [16], the maintenance of acceptance tests is an issue that appears from the very start of a project. The tests need to be constantly maintained and the test suite should be integrated in a continuous build process [1].

In the agile development it is desirable that users participate in the specification of the acceptance tests, while the testers develop the test automation. It is assumed that a good specification of acceptance tests facilitates the communication among the stakeholders, helping the test automation by the testers. Moreover, specification that involves multiple users can increase the complexity, mainly when users adopt particular data for the tests. For example, the data of an end-of-session requirement can be called "Log out" by one user and "Sign out" by another one. Both data of end-of-session might communicate the intent to the tester, however the ambiguity can lead the tester to a misunderstanding in a way that it is implemented distinct glue code for the same requirement. In this case it is important to uniform the test data before automating the test in order to avoid unnecessary glue code implementation. In [11], Longo and Vilain proposed a metric to measure the data uniformity of acceptance tests in order to detect exaggerated variances in data that may weaken the quality of acceptance tests.

To investigate the impact of data uniformity in automated acceptance tests, in this work we performed an experiment using the User Scenarios through User Interaction Diagrams (US-UIDs) format for the acceptance tests. To execute US-UIDs as acceptance tests, it is necessary to bind them to the SUT. The binding between the US-UIDs and the SUT is done through the fixture names and the glue code [10]. Fixture names are labels created a-posteriori of the US-UIDs specifications to bind the US-UID elements to the glue code. The glue code is responsible for exchanging information between the US-UIDs and the SUT during the test execution. While the specification of the US-UIDs representing the acceptance testes is a responsibility of the clients and users, the fixture naming and glue code creation is a responsibility of the testers [10].

This paper is organized as follow. Section 2 shows details of the problem of data uniformity in the automation of acceptance tests represented by US-UIDs. Section 3 describes the experiment performed and the evaluation methodology. Section 4 presents the results. Finally, Section 5 shows the conclusions.

II. EXAMPLE OF THE PROBLEM

The User Scenarios through User Interaction Diagrams (US-UIDs) are used for software requirements specifications [8, 9, 10]. These diagrams can be used as automated acceptance tests. The US-UIDs have been suggested as a specialization of the technique UID [20], where the information is replaced with concrete values from the user scenarios. The applicability of the US-UIDs is usually made by non-technical users to create acceptance testing before the development.

In the following we talk about the problem of data uniformity in the acceptance tests represented by US-UIDs. To illustrate this problem, Figure 1 shows a pair of US-UIDs. The pair of US-UIDs shown in this example are for specifying requirements of an application similar to WhatsApp. The US-UID A refers to the requirement to respond to a message, and US-UID B refers to the requirement to check status. The example considers two US-UIDs with the objective of showing data uniformity problems only in first interaction state (represented by ellipses [8]). The uniformity is calculated comparing data from user inputs (represented by rectangle with string inside [8]) and system outputs (loose strings [8]) between the US-UIDs.

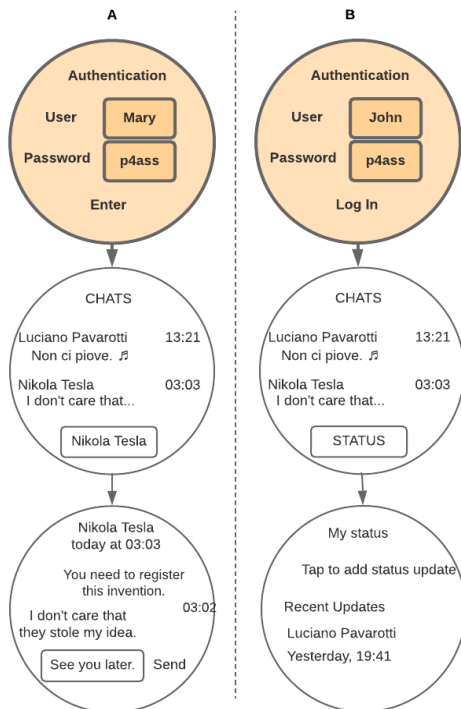


Figure 1. Two US-UIDs with problems of uniformity in data.

The example highlights two problems of data uniformity, where the first one is related to the values of the user inputs “Mary” and “John”. These two values are different from each other, but, still, have the same meaning. The meaning is clearly a username. Specially, in US-UIDs with features little known

among the stakeholders, this type of non-uniform data might cause loss of meaning leading to miss understandings.

The second problem of data uniformity is related to the system outputs “Enter” (US-UID A, Figure 1) and “Log In” (US-UID B, Figure 1). Both system outputs represent, in the SUT implementation, the text of the button for the action for system access. For communication purposes, the stakeholders are able to understand that the text in these two system outputs which, despite the difference, have the same meaning. However, using these US-UIDs as automated tests leads to an unnecessary implementation of the glue code, because the glue code must provide two distinct values (“Enter” and “Log in”) for the same requirement.

For the example of the US-UIDs from Figure 1 (considering only the first state of interaction of each US-UID and non-uniform data), three strategies for fixture naming can be developed. These strategies impact in different glue code implementations and also spreads the problems of data uniformity in different glue code structures. The Figure 2 shows three different strategies to naming fixtures and Figure 3 shows de glue code for each strategy.

A. Strategy of Shared Fixture Names (S1)

The strategy of shared fixture names causes two interaction states (Figure 2, S1) and two user inputs or two system outputs from different US-UIDs share the same fixture name. Figure 2 shows three possible shared fixture names: “AuthenticationState” for the two interaction states, “user” for two user inputs, and “enterButton” for two system outputs. The two interaction states of the US-UIDs from the example are named by “AuthenticationState”, sharing, thus, the same fixture name.

The fixture name “user” is used to name the user inputs “Mary” and “John” which represent the user names. Although this fixture name is not the same as the input names, there is no sense of conflict with these names for the example. However, the fixture name “enterButton” for the system outputs “Enter” and “Log In” might create doubts. The proper decision of the developer is to return the US-UIDs to the users or clients and solve which system output name is the most appropriate. However, if the developer keeps the names irregular, the glue code will be developed as shown in Figure 3 (S1).

During the creation of the glue code, it is necessary to implement the class for the interaction states and the methods for each user input and system output in order to bind the US-UIDs to the SUT. The interaction states are linked to the class “AuthenticationState”. The method `getEnterButton` is responsible to bind the SUT to the fixture “enterButton” and to return the result to fulfill the test exercise. In line 6 (S1, Figure 3) the SUT is represented only by the string “Enter OR Log In?”. In this case a conflict is generated (Figure 3, S1, line 6) because the method may simply return one value, but in US-UIDs there are as candidates the values “Enter” or “Log In”, in which, this way, the irregular data is detected due to the conflict on the decision of method return. Therefore, the developer will have a lag upon reviewing the US-UIDs. The reviewing process can take longer and increase the costs if help from users or clients is needed.

B. Strategy of Fixture Names with Hybrid Sharing (S2)

The strategy of fixture names with hybrid sharing occurs when two or more interaction states share the same fixture name, and two user inputs or system outputs have different fixture names. Figure 2 (S1) shows the possible fixture names that show a problem of uniformity and both interaction states from US-UIDs A and B share the same glue code class. The user inputs “Mary” and “John” have problems of data uniformity but, in practice, they are easily detected. Therefore, in the example of strategy of fixture names with hybrid sharing, the user inputs are not considered. As an example, both interaction states share the fixture “*AuthenticationState*” and the fixture names “*enterButton*” and “*loginButton*” are fixture names that are not shared for the system outputs “Enter” and “Log In”. Therefore, the system outputs with similar meaning are handled in different ways. Figure 3 (S2) shows the glue code for the fixture name of hybrid sharing for the example from Figure 2 (S2).

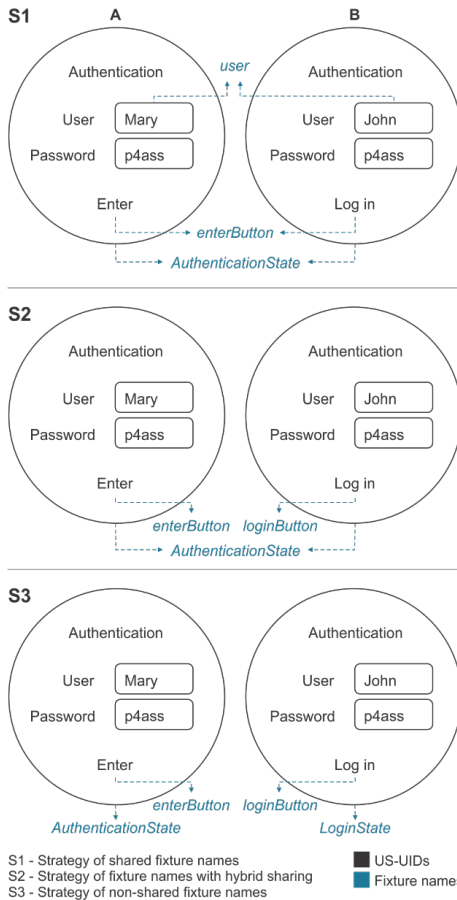


Figure 2. Example of three strategies for fixture naming can be developed.

The fixture code allows the use of a class with non-shared methods, therefore, distinct methods allow returns of different values. The method *getEnterButton* is responsible for binding the SUT to the fixture “*enterButton*” and for returning the result to fulfill the test exercise. The method *getLoginButton* is responsible for binding the SUT to the fixture “*loginButton*” and for returning the value that fulfills the test exercise. Therefore, both methods have different responsibilities and do not cause any conflicts. It is important to point out that, in this example, both methods return the values from test results just because they are not linked to the SUT yet.

C. Strategy of Non-Shared Fixture Names (S3)

The strategy of non-shared fixture names occurs when two interaction states to have different fixture names, as well as two user inputs or two system outputs to have different fixtures. Figure 2 (S3) demonstrates the possible fixture names that show a problem of uniformity.

```

----- S1 -----
1 @Fixture("AuthenticationState")
2 public class AuthenticationFixture {
3
4     public void setUser(String name) {}
5     public String getEnterButton() {
6         return "Enter OR Log In?";
7     }
8 }
/* The return value of the method is conflicting for the two US-UIDs.
When the programmer arrives at this point it is common to detect the
conflict. */

----- S2 -----
1 @Fixture("AuthenticationState")
2 public class AuthenticationFixture {
3
4     public String getEnterButton() {
5         return "Enter";
6     }
7     public String getLoginButton() {
8         return "Log In";
9     }
10 }
/* For this type of fixture sharing the problem of data uniformity can be
detected only if we analyze the two methods. */

----- S3 -----
1 @Fixture("AuthenticationState")
2 public class AuthenticationFixture {
3
4     public String getEnterButton() {
5         return "Enter";
6     }
7 }
8 @Fixture("LoginState")
9 public class LoginStateFixture {
10
11     public String getLoginButton() {
12         return "Log In";
13     }
14 }
/* In this strategy, the data uniformity problem can be detected only if
you parse the two methods, but note that they are in different classes. */

```

Figure 3. Glue code for the three examples of Figure 2.

In this example we are only considering both interaction states and system outputs “*Enter*” and “*Log In*”, so there is no fixture name sharing. This way, the interaction states and system outputs with similar meaning are treated in different ways. Figure 3 (S3) shows the glue code for the non-shared fixture names for the example from Figure 2 (S3).

In the level of fixture code, both US-UIDs use two different classes for binding the US-UIDs to the SUT, thus classes and methods are not shared. The class *AuthenticationFixture* (Figure 2, S3, line 2) contains the methods for the interaction state of US-UID A (Figure 2, S3, left). For the interaction state of US-UID B (Figure 2, S3, right) the class *LoginState* (Figure 2, S3, line 10) is used. This way, the methods for binding the system

outputs “Enter” and “Log In” to the SUT are allocated in different classes. The method *getEnterButton* is responsible for binding the SUT to the fixture “*enterButton*” and for returning the result to fulfill the test exercise. The method *getLoginButton* is responsible for binding the SUT to the fixture “*loginButton*” and for returning the result that fulfills the test exercise. Therefore, both methods have different responsibilities and do not cause any conflict. Again, in this example, both methods return the values from test results because they are not linked to the SUT yet.

III. EXPERIMENT

The experiment evaluates the treatment of data uniformity in the automation process of acceptance tests using the US-UIDs format. In the experiment, activities such as US-UIDs specification, fixtures naming, and glue code creation are elaborated. The purpose is to investigate the following research questions:

- RQ1: Does the help of an expert in the US-UIDs specification improve the data uniformity?
- RQ2: Does the treatment of uniformity affect any of the strategies of fixture name sharing?
- RQ3: How long does the fixture naming take? How much of the volume of glue code is implemented?

A. Methodology for Evaluation

For the evaluation, a specification of US-UIDs from requirements of a messaging system has been considered. Figure 4 shows an activities diagram with the three steps of the evaluation: preparation, experiment and result analysis.

1) *Preparation*. In the preparation step, the participants are trained on the language of the US-UIDs for about 15 minutes; this training is the same as the adopted by Longo and Vilain [8, 9]. During the preparation, non-technical participants are oriented to use pencil, rubber and paper for a small training. After the training with the US-UID, explanations on the messaging system are given. The participants are requested to figure out requisites for an application similar to WhatsApp, Telegram, Hangout or Messenger and, then, specify the requisites as US-UIDs.

2) *Experiment*: In order to answer the questions and to analyze the treatment of data uniformity in the specification of US-UIDs by non-technical¹ participants, the specifications are divided in two projects (P1 and P2). In the project P1, the specification is done by one participant. After the specifying, an expert² checks the data uniformity for each US-UID. If the US-UID does not show data uniformity, it is then returned to the participant so that he may improve the uniformity according to the expert guidelines. If, in the evaluation from an expert, the US-UID shows a good data uniformity, then the US-UID specification is finished. In the project P2, the specification is done by four non-technical participants without the expert's

help. After the specification step is finished, the activity of fixture naming is stated, which is carried out by only one tester. The time of this activity is timed, as it also shows the tester effort to understand the US-UIDs. For all the US-UIDs, the fixtures are named, and this information is necessary for binding the US-UIDS elements to the glue code. After the activity of fixture naming is finished, the activity of glue code creation is carried out. An algorithm has been implemented in order to generate the glue code. The algorithm has the US-UIDs as an input and generates the glue code according to the framework proposed by Longo et al. [10].

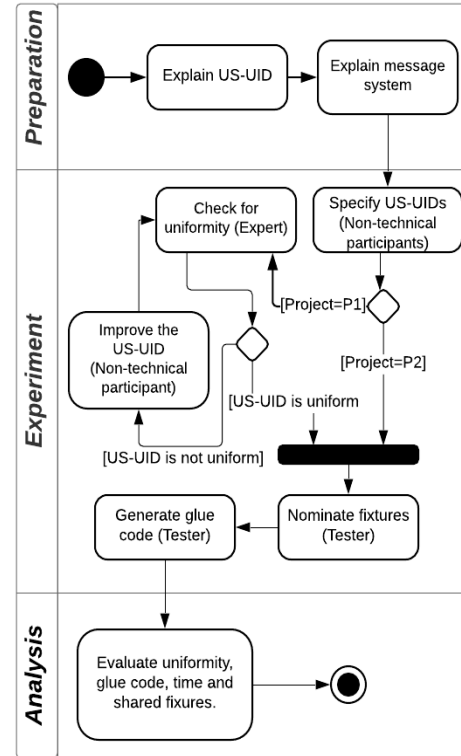


Figure 4. Activity diagram for uniformity evaluation.

IV. RESULTS

A. Descriptive Measures of the US-UIDs

In calculating the measures, all the US-UIDs specified by the participants have been added. This way, there are differences in the descriptive measures of US-UIDs that specify the same requirement, as the specification way of each participant is different. In the project P1, 6 US-UIDs, 31 interaction states, 126 system outputs, 43 user inputs and 25 transitions have been specified. Whereas in project the P2, 20 US-UIDs, 59 interaction states, 308 system outputs, 66 user inputs and 39 interaction states have been specified. There are no doubts that differences exist in the number of elements of the US-UIDs, but it highlights that the intention is specifying the same requirement in both projects.

¹ Non-technical participants are individuals without technical training in information technology.

² Expert is an experienced developer with US-UIDs.

B. Data Uniformity of the US-UIDs (RQ1)

In order to evaluate the data uniformity between both projects, a metric of data uniformity proposed by Longo and Vilain [11], is used. The metric compares pairs of US-UIDs and the results in a quantitative value. The value measured is in the interval [0%, 100%], where 0% indicates total irregularity and 100% indicates total uniformity. In order to apply the metrics, the US-UIDs resulted from experiment, and that were described on paper, are scanned to the tool Sc3n4r10³ and, then, the metric is applied computationally. Question RQ1 can be answered by the following formula with the following hypothesis:

$$H0: Uniformity_{p1} = Uniformity_{p2}$$

$$H1: Uniformity_{p1} \neq Uniformity_{p2}$$

$Uniformity_{p1}$ is the uniformity measure of the US-UIDs from project P1; $Uniformity_{p2}$ is the uniformity measure of the US-UIDs from project P2. The decision for accepting H0 or H1 is made according to the data collected from the experiment. The decision for accepting the hypothesis H0 means that the adding of the treatment of uniformity by the expert has no effect in the specification. However, when it accepts the hypothesis H1, it states that the uniformity is different between both projects, thus it is necessary to compare the uniformity averages to discover which one is the best.

The distribution of the data uniformity was analyzed through a Z-Test, at a level of significance of ($\alpha = 0.05$), obtaining a meaningful statistic difference among the uniformity of both projects ($P_{value}=2.2e^{-16}$). Therefore, then with the inclusion of the expert, there is significant difference in the uniformity of the US-UIDs data in both projects (H1).

Figure 5 shows the data uniformity in both projects. Project P1 generated 30 pairs of US-UIDs with uniformity average of 64%, median of 67% of uniformity, and uniformity interval between [32%, 100%]. Project P2 generated 380 pairs of US-UIDs with uniformity average of 12%, median of 6% of uniformity, and uniformity interval between [0%, 42%], with some discrepant values above this interval. The discrepant values occurred by US-UIDs where the participant had noticed the importance of keeping data uniformity in order to better specify the US-UIDs.

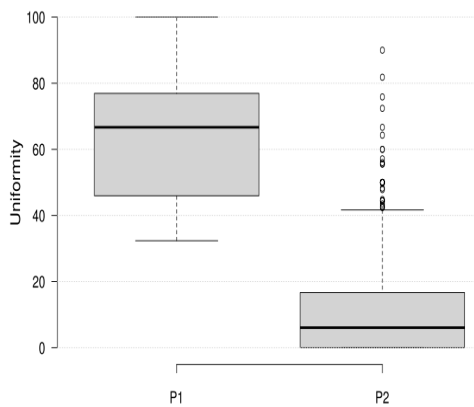


Figure 5. Uniformity of each project per pair of US-UIDs.

C. Fixture Name Sharing (RQ2)

The strategies of fixture name sharing were measured by the times that each fixture name is shared, sectioned by interaction states, system outputs and user inputs. For example, according to Figure 2 (S1), the fixture name “user” is shared two times, i.e., “user” is shared by the user inputs “Mary” and “John.” For the system outputs and interaction states the count is the same. The count was applied to each project with the aid of an algorithm implemented together with tool Sc3n4r10. Question RQ2 can be answered by a hypothesis test about the number of elements that share the same fixture name with the following hypothesis:

$$H0: FixtureNameSharing_{p1} = FixtureNameSharing_{p2}$$

$$H1: FixtureNameSharing_{p1} \neq FixtureNameSharing_{p2}$$

$FixtureNameSharing_{p1}$ is the number of times each fixture name is shared in US-UIDs from project P1; $FixtureNameSharing_{p2}$ is the number of times each fixture name is shared from project P2.

The hypothesis can be formulated for fixture name shares of user inputs, system outputs, and interaction states. The decision for accepting H0 means that the help of an expert has no effect in the strategies of fixture sharing adopted by the tester. However, by accepting the hypothesis H1, it is affirmed that the strategies of fixture sharing adopted by the tester are different between both projects. Table 1 shows the statistics analysis about the fixture name sharing for each element type. For each element type, the statistic tests shown suggest that there is no meaningful difference in the fixture sharing between projects P1 and P2. Therefore, the treatment of uniformity does not affect the strategies of fixture sharing for all element type (H0).

TABLE I. STATISTICAL ANALYSIS.

Elements	Test	P _{value}	Statistical Decision ($\alpha = 0.05$)
Interactions	U Test	0.1301	H0
Outputs	Z Test	0.4426	H0
Inputs	U Test	0.0685	H0

D. Effort (RQ3)

The effort for fixture naming by the tester is analyzed through time data. Therefore, it is necessary to compare the time spent in each project. The glue code volume also indicates indirectly the effort of the activity of fixture naming.

Figure 6 shows the glue code volume and the time spent in the activity of fixture naming by the tester. Project P1 resulted in 255 lines of code and took 240 minutes to name the fixtures. Project P2 resulted in 834 lines of code and took 552 minutes to name the fixture.

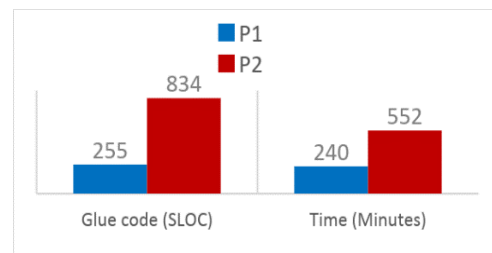


Figure 6. Measures of code volume and time.

³ <https://github.com/douglashiura/us-uid.git>

V. CONCLUSIONS

This paper proposes the treatment of data uniformity in the activity of specifications of the US-UIDs and evaluates the effect of this data uniformity in the glue code reuse. In the evaluation, the treatment of uniformity was done by a participant who is an expert during the activities where non-technical participants specify the US-UIDs. In fact, the result of the treatment by the expert increase the uniformity, i.e., there is a meaningful statistic difference between the project with the treatment of uniformity done by the expert and a project without the same. However, in the activity of fixture naming, developed by the tester, the difference of data uniformity does not affect the fixture name sharing, i.e., the uniformity is not a factor that inclines the tester in the choosing of strategies of fixture name sharing. The developing time of the activity by the tester is very different between a project with uniformity treatment and a project without treatment. Although the treatment of uniformity does not show any influence in the strategies of fixture name sharing, the tester spends more time understanding and naming the fixture in a proper way; the time difference is twice more for a project that does not treat the data uniformity. The glue code volume is also very different. A higher code volume indicates less reuse and more effort in the activity of maintenance. Thus, it is concluded that data uniformity should be addressed at the time of US-UIDs specification, as these later saves time for the naming of fixtures and significantly reduces the amount of glue code.

As a threats of validity it is important to emphasize that this work considered only the initial requirements of a messaging system, so the results may be different for a scope of applications with lots of requirements. However, it is important to maintain uniformity in project implementation because the tests are more comprehensive, which impacts in less time to develop them (although there is interaction between developers and customers for uniformization) and less glue code to maintain during development. Yet, the software development challenges from the real world, many times, rely on projects with more stakeholders, that way we need investigate the problem in projects with different amounts of participants.

Other works address the quality improvement on specifying user stories. Lucassen et al. [13, 14] proposed a framework to evaluate fourteen quality criteria. These criteria are adopted in the Grimm Method [15] and applied in a case of study in the industry. However, the applying of the Grimm Method did not result in a meaningful difference in the software development process.

Finally, the reliability of an experiment refers to the capacity of other researchers to reply the methodology [12]. For the replication, the methodology has been detailed and algorithms for uniformity measuring and fixture sharing along with the tool Sc3n4r10 have been made available. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) - Finance code 001.

REFERENCES

- [1] B. Haugset and G. K. Hanssen, Automated acceptance testing: A literature review and an industrial case study. Agile, 2008. AGILE'08. Conference, IEEE, pp. 27-38, 2008.
- [2] T. Dybå and T. Dingsøyr, Empirical studies of agile software development: A systematic review. Information and software technology. Elsevier, vol. 50, pp. 833-859, 2008.
- [3] G. K. Hanssen and B. Haugset, "Automated acceptance testing using fit" in System Sciences, HICSS'09, 42nd Hawaii International Conference on. IEEE, pp. 1-8, 2009.
- [4] Cohn, Mike. User stories applied: For agile software development. Addison-Wesley Professional, 2004.
- [5] Wautelet, Y., Heng, S., Kolp, M., & Mirbel, I. "Unifying and extending user story models" in International Conference on Advanced Information Systems Engineering. Springer, Cham, pp. 211-225, 2014.
- [6] Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., & Brinkkemper, S. "The use and effectiveness of user stories in practice". in International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, Cham. p. 205-222, 2016.
- [7] Druk, Michael, and Martin Kropp. "ReFit: A Fit test maintenance plug-in for the Eclipse refactoring plug-in." in Developing Tools as Plug-ins (TOPI), 2013 3rd International Workshop on. IEEE, pp. 7-12, 2013.
- [8] D. H Longo., and P. Vilain. User scenarios through user interaction diagrams. *International Journal of Software Engineering and Knowledge Engineering* 25.09n10, pp. 1771-1775, 2015.
- [9] D. H Longo., and P. Vilain. Creating User Scenarios through User Interaction Diagrams by Non-Technical Customers. in Software Engineering and Knowledge Engineering – SEKE15. KSI Research Inc. and Knowledge Systems Institute Graduate School, pp.330-335, 2015.
- [10] Longo, D. H., Vilain, P., da Silva, L. P., and Mello, R. D. S. A web framework for test automation: user scenarios through user interaction diagrams. in Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services. ACM, pp 458-467, 2016.
- [11] D. H Longo., and P. Vilain. Metrics for Data Uniformity of User Scenarios through User Interaction Diagrams". Software Engineering and Knowledge Engineering – SEKE18. KSI Research Inc. and Knowledge Systems Institute Graduate School, pp.1-6, 2018.
- [12] A. K. Massey, R. L. Rutledge, A. I. Anton, P. P. Swire. Identifying and classifying ambiguity for regulatory requirements, " Requirements Engineering Conference (RE), 2014 IEEE 22nd International, IEEE, pp. 83 – 92, 2014.
- [13] Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., and Brinkkemper, S. Improving agile requirements: the quality user story framework and tool. *Requirements Engineering*, v. 21, n. 3, p. 383-403, 2016.
- [14] Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., and Brinkkemper, S. Forging high-quality user stories: towards a discipline for agile requirements. in Requirements Engineering Conference (RE), 2015 IEEE 23rd International. IEEE. p. 126-135, 2015.
- [15] Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., and Brinkkemper, S. Improving user story practice with the Grimm Method: A multiple case study in the software industry. in International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, Cham, 2017.
- [16] Borg, Rodrick; Kropp, Martin, "Automated acceptance test refactoring". in Proceedings of the 4th Workshop on Refactoring Tools. ACM. p. 15-21, 2011.
- [17] Dos Santos, E. C.; P. Vilain, "Automated Acceptance Tests as Software Requirements: An Experiment to Compare the Applicability of Fit Tables and Gherkin Language". In. International Conference on Agile Software Development. Springer, Cham. p. 104-119, 2018.
- [18] Sommerville, I.: Software Engineering. 9th edn. Pearson Education, Boston 2015.
- [19] Torchiano, M., Ricca, F., M. D. Penta, "Talking tests": a preliminary experimental study on fit user acceptance tests. in First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), p. 464-466, 2007.
- [20] Zeferino, N. V., and Vilain, P. "A model-driven approach for generating interfaces from user interaction diagrams" in Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services. ACM, 474-478, 2014.

Test Case Generation by EFSM Extracted from UML Sequence Diagrams

Mauricio Rocha^{1,2}, Adenilso Simão¹, Thiago Sousa², Marcelo Batista²

¹*Instituto de Ciências Matemáticas e de Computação (ICMC), USP, São Carlos, SP, Brazil*

mauriciormrocha@usp.br, adenilso@icmc.usp.br

²*Centro de Tecnologia e Urbanismo (CTU), UESPI, Teresina, PI, Brazil*

mauricio@ctu.uespi.br, thiago@ctu.uespi.br, marcelo.araujo@uespi.br

Abstract—The effectiveness of Model-Based Testing (MBT) is mainly due to the potential for automation it offers. If the model is formal and machine-readable, test cases can be derived automatically. The Extended Finite State Machine (EFSM) is a formal modeling technique widely used to represent a system. However, EFSM is not a common practice in industry. On the other hand, the Unified Modeling Language (UML) has become the de-facto standard for modeling software, but due to the lack of formal semantics, its diagrams can have ambiguous interpretations and are not suitable for testing automation. In this context, we present a systematic procedure for generating tests from a UML model. More specifically, our approach proposes a mapping from the UML Sequence Diagram into Extended Finite State Machine in order to provide a precise semantics to them and uses the ModelJUnit and JUnit libraries in order to generate test cases automatically.

Index Terms—Model-Based Testing, Model-Driven Engineering, Sequence Diagram, Extended Finite State Machine, ModelJUnit, JUnit

I. INTRODUCTION

A common practice in most software development processes is the use of abstract models to aid in the construction of products. These models represent the essential parts of a system and allow software engineers to take a conceptual view of several different software perspectives. An option for software modeling is the Unified Modeling Language (UML), since it is widely used and, due to its expressiveness, it is possible to model both static and structural aspects as well as dynamic or behavioral [1]. However, due to the lack of formal semantics, the use of UML can lead to some issues, such as inconsistency, transformation problems and different interpretations [2].

An option to minimize these problems is the use of formal models, since they have a precise semantics to accurately represent system behavior. However, what is observed in practice is that formal methods are little used in industry, probably due to the lack of training and familiarity with the mathematical notation by the developers.

In the context of software testing, modeling can increase the productivity of this activity. According to Utting et al. [3], Model-Based Testing (MBT) allows the automatic generation of tests from models and other software artifacts, making it possible to create tests for the software even before coding,

thus reducing the cost of development. The central idea of MBT is generating input sequences and their expected outputs from a model or specification. The input sequences are then applied to the System Under Test (SUT) and the software outputs are compared to the outputs of the model. This implies that the model must be valid, i.e. faithfully represent the requirements. Basically, MBT are used for functional black-box testing, where software functionality is examined without any knowledge of the software's internal coding.

In MBT, it is recommended to use formal models, since they can be used as a basis for automating the testing process, making it more efficient and effective [4]. There are several formal modeling techniques based on state transition machines that can be used to specify a test model. Extended Finite State Machine (EFSM) has been widely used in the formal methods community, since they make it possible to represent the flow of control and data of complex systems. Moreover, EFSM can be implemented as a test model using the the ModelJUnit [5] library, which was designed as an extension of JUnit. Therefore, the models are written in Java, a popular programming language.

In this context, we present a systematic procedure for generation of test cases from a UML model. The idea is to use concepts of Model-Driven Engineering (MDE) to transform the UML Sequence Diagrams into EFSM and using the ModelJUnit and JUnit libraries in order to generate test cases automatically. In summary, the main contributions of this paper include:

- 1) Definition of transformation rules for mapping the elements of the UML Sequence Diagram into the Extended Finite State Machine constructions using Atlas Transformation Language (ATL) [6].
- 2) Formalization of the UML Sequence Diagram into EFSM which is a semantically accurate model.
- 3) Automatic source code generation of ModelJUnit and JUnit classes from EFSM using Acceleo [7].
- 4) Systematic procedure to generate Java tests from UML Sequence Diagram automatically.

II. BACKGROUND

A. Sequence Diagram

The dynamic behavioral aspect of an object-oriented software is defined through the interaction of objects and the

exchange of messages among them. The main diagram of the interaction model is the UML Sequence Diagram, which presents the interactions between objects in the temporal order in which they occur.

Lifelines represent participants of the interaction that communicate via messages. These messages may correspond to the operation call, signal sending or a return message. More complex interactions can be created using combined fragment. A combined fragment is used to define control flow in the interaction. It can be composed of one or more operands, zero or more interaction constraints, and an interaction operator. An operand corresponds to a sequence of messages that are executed only under specific circumstances. Interaction constraints are also known as guard conditions and represent a conditional expression.

In this paper, we use three interaction operators that model the main procedural constructs:

- **alt**: construction of the if-then-else type. Only one operand will be executed.
- **opt**: construction of the if-then type. It is very similar to the alt operator, with the difference being that only one operand is defined, which may or may not be executed.
- **loop**: a construct that represents a loop where the single operand is executed zero or more times.

Other interaction operators defined by UML 2, that can be found in OMG (Object Management Group) [1], are not in the scope of this work.

B. Model-Driven Transformation

Model transformation is a key concept within the scope of Model-Driven Engineering (MDE). The MDE aims at supporting the development of complex software that involves different technologies and application domains, focusing on models and model transformation [8].

Similarly to models, metamodels play a key role on the MDE. A metamodel makes statements about what can be expressed in valid models of a given modeling language. Modeling languages need to have formal definitions so that transformation tools can automatically transform the models built into those languages. The OMG has created a special language called Meta Object Facility (MOF) [9], which is the default metalanguage for all modeling languages. Thus, each language is defined by means of a metamodel using the MOF.

Model transformation is the generation of a target model from a source model. This generation process consists of a set of transformation rules that describes how the elements of the source model are mapped into elements of the target model. The transformations can be performed in two ways: Model-To-Model (M2M) mapping or Model-To-Text (M2T) mapping.

C. Extended Finite State Machine

An Extended Finite State Machine (EFSM) consist of states, predicates, and assignments related to variables between transitions, so that it can represent the control and data flow of complex systems.

An EFSM can be formally represented by a 6-tuple (s_0, S, V, I, O, T) [10], where:

- S is a finite set of states with the initial state s_0 ;
- V is a finite set of context variables;
- I is a set of transitions entries;
- O is a set of transitions outputs;
- T is a finite set of transitions.

Each transition $tx \in T$ can also be represented formally by a tuple $tx = (s_i, s_j, P_{tx}, A_{tx}, i_{tx}, o_{tx})$, where s_i, s_j and $i_{tx} \in I$ represents the input parameters of the beginning of the state transition tx and $o_{tx} \in O$ represents the output parameter at the end of the state transition tx . In addition, P_{tx} represents the predicate conditions (guards) with their respective context variables and A_{tx} the operators (actions) with their respective current variables.

D. Model-Based Testing

The software test aims to perform an implementation of the system under construction with test data and verify that its operating behavior conforms to its specification. This implementation being tested is named the System Under Test (SUT).

In MBT, the use of models is motivated by the observation that, traditionally, the testing process is unstructured, non-reproducible, undocumented and depends on the creativity of software engineers. The idea is that artifacts used in SUT coding can help mitigate these problems [3].

In summary, the MBT covers the processes and techniques for automatic derivation of test cases from abstract software models. To achieve success in this activity, rigor is necessary in this process.

III. OUR APPROACH

In this section we present a systematic process for test case generation by EFSM extracted from UML Sequence Diagrams. The Figure 1 illustrates our approach, which is divided into two main steps as detailed below:

Step 1 - Transformation between models. Scenarios are written in the form of the UML Sequence Diagram. This UML Sequence Diagram is transformed into an EFSM through the mapping between their respective metamodels using Atlas Transformation Language (ATL). The result of this step is a formal software model represented by an EFSM.

Step 2 - Generation of test cases. From a model of the software represented by EFSM, the test cases are generated using EFSM-based test generation methods from ModelJUnit and JUnit libraries. In this step, a Model-To-Text (M2T) transformation is performed using Acceleo, resulting in a set of test cases.

A. Metamodels

We define the UML Sequence Diagram metamodel (source) and Extended Finite State Machine metamodel (target). These metamodels were implemented in Ecore using the Eclipse Modeling Framework (EMF) [11].

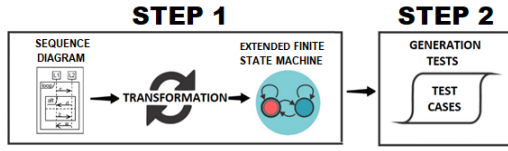


Fig. 1. Our Approach.

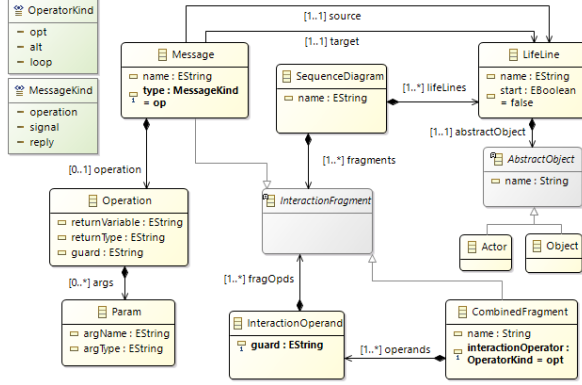


Fig. 2. Sequence Diagram Metamodel.

The complete official UML specification [1] is very complex because abstract syntax is represented in several separate diagrams, which makes it difficult to see all the connections between the important elements. In addition, the specification uses the so-called semantic variation points, meaning part of the semantics is not specified in detail to allow the use of the UML in many domains. Therefore, the official UML metamodel is heavily criticized for having many elements that are seldom used in practice [12], [13]. In this scenario, the metamodel presented in Figure 2 is simpler than the one specified by the OMG for the Sequence Diagram, and does not have constructs that are rarely used in practice. The metamodel proposed contains 13 metaclasses. The use of simplified metamodels occurs in most of the papers published in the literature [14], [15], [16].

The proposed metamodel for EFSM presented in the Figure 3 is based on the formal definition of Yang et al. [10] explained in section II.C. The metamodel is composed of six metaclasses, among which, EFSM represents an Extended Finite State Machine. The EFSM entity is composed of states, transitions and context variable.

B. Transformation Rules

In this section, we present the transformation rules between the Sequence Diagram and the Extended Finite State Machine. The following transformation rules have been defined:

- **InitFsm:** this rule creates an EFSM with the name of the sequence diagram and adds the initial state S_0 . The previous state and the current state are updated with the initial state. This rule can only be applied once.
- **Transition:** for all messages of type signal ($type = si$) or operation ($type = op$), a state is added (which is now the

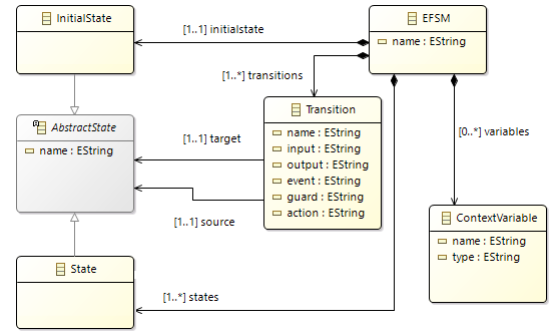


Fig. 3. Extended Finite State Machine Metamodel.

current state), and a transition that connects the previous state to the current state of the EFSM. The input for this transition will be labeled with the name of the message. If the message operation has a return, the output, guard, and action of this transition are labeled with the return of the operation. In addition, the event is labeled with the name of the operation, its return, and its arguments.

- **ContextVariable:** for all messages of type operation ($type = op$) that have a different return of *void*, a context variable is created with the name and return labeled with the name and return of the operation.
- **Alt and Opt:** when a fragment combined with the *alt* operator or *opt* operator is found in Sequence Diagram, it is added a new state for each operand and a new transition linking the current state to each of the created states. Every transition will have its input labeled with the message name and its output labeled with the guard of the respective operand.
- **Loop:** when a fragment combined with *loop* interaction operator is found in the Sequence Diagram, a reply message must be defined as the last message of the snippet. As soon as the process finds this message, a new state (which is now the current state) and a transition that connects the previous state to the current state in the EFSM are added. The input of this transition will be labeled with the name of the message and the output with the negation of the operator's guard. Another transition is created by connecting the previous state to the last state created before fragment. The input of this transition will be labeled with the name of the message and the output with the guard of the operator.

In our approach, these transformation rules were implemented using Atlas Transformation Language (ATL). ATL is one of the packages developed in the AMMA (ATLAS Model Management Architecture) model engineering platform [6]. ATL rules may be specified either in a declarative style (*Matched Rules*) or in an imperative style (*Called Rules*). *Lazy Rules* is kinds of *Matched Rules* are triggered by other rules.

In order to make feasible the transformations described above, the following *lazy rules* were implemented:

- **LrInitialState:** creates the initial state S_0 , increments the

order of the states, and changes the previous state and the current state as the initial state created. In addition, the name of the Sequence Diagram being scanned is saved in a variable.

- **LrState**: creates a new state, increments the order of states, the previous state is changed to the current state and the current state changed to the new created state.
- **LrTransition**: creates a transition that connects the previous state to the current state. The transition input and event are labeled with Sequence Diagram message information. The output, guard, and action can be null and depend on the operator and message type of the Sequence Diagram.
- **LrContextVariable**: this rule creates a context variable with the name and type labeled with the return variable of the operation and the type of the operation retract, respectively.

All of these rules implemented in ATL are available in the *Transformation/SD2EFSM/SequenceDiagram2EFSM.atl* file of the approach [repository](#)¹.

C. Test Case Generation

For test case generation our approach uses the ModelJUnit and JUnit libraries, since they are open-source and their uses are very simple for Java programmers. In addition, the ModelJUnit library enables the implementation of formal models widely used in MBT, such as EFSM. Other advantages of using ModelJUnit is that it provides a variety of useful test generation algorithms, model visualization features, model coverage statistics, and other features [17].

The process of implementing the MBT environment in the ModelJUnit and Junit libraries consists of four steps:

- 1) **The Model**: initially, we have implemented the *Fsm-Model* interface to define our model in ModelJUnit. In this Java Class we define in a enumeration variable (*enum State*) all the possible states of our EFSM and for each context variable we define a variable in the class. For each input in our model, we wrote action methods (*@Action*) to define the transitions that link the states of our model. In addition to these methods, we define in our model the *getState* method that returns the current state and the *reset* method that takes the machine to the initial state.
- 2) **The Adapter**: in this step we implemented the Adapter class that allows our model to communicate with and take control of our SUT. For each one of the action method define in the model that trigger an event, we added a similarly named method in the adapter class. In our model defined in Step 1, we call the correct adapter method in each action method. In addition, in the Adapter class we need to instantiate an object for each class of the SUT.
- 3) **Generation Tests**: in this step, we initially need to instantiate the model defined in Step 1. Then, we have

to choose the test strategy that will be used. ModelJUnit offers four different strategies: AllRoundTester, GreedyTester, LookaheadTester and RandomTester. In our approach we used the LookaheadTester test strategy, since it is a more sophisticated algorithm and can cover all transitions and states quickly [17]. Finally, we call the *buildGraph* method to build the graph and generate the tests. This graph will also be used to calculate coverage metrics for transitions, states, and action.

- 4) **Test Concretization**: In this step, the test cases were implemented in Java using the JUnit library.

These four steps described above were automatically generated by Model-To-Text (M2T) transformation using Aceleo. Aceleo is a template-based technology including authoring tools to create custom code generators. It allows you to automatically produce any kind of source code from any data source available in EMF format [7]. We have implemented the *generateClassModel*, *generateClassAdapter*, *generateClassTest* and *generateClassJUnit* generators modules. The input of these modules is the EFSM generated in step 1 of our approach.

These code generators implemented in Aceleo are available in the *Transformation/Efsm2ModelJUnit/src/Common/* folder of the approach [repository](#)¹.

IV. EXAMPLE

In this section, we use an example to illustrate the application of our approach. The UML Sequence Diagram of Figure 4 presents interactions of an ATM (Automatic Teller Machine) for the withdrawal scenario.

Initially, using the Sequence Diagram editor implemented in the EMF, we created the Sequence Diagram model described in Figure 4. Then, using the transformation rules implemented in ATL, the UML Sequence Diagram is converted into an Extended Finite State Machine. At the end of the execution of the transformation rules we will have an EFSM as shown in Figure 5.

In Step 2 of our approach, from the EFSM extracted in Step 1, the test cases are generated. Using the generator modules implemented in Aceleo, the classes (*AtmModel*, *AtmAdapter*, *AtmTest* and *AtmJUnit*) are generated automatically.

The *AtmModel* class is an implementation of the *FsmModel* interface. In this class is defined the variable enumeration *State* that represents all the states (*S0*, *S1*, *S2*, *S3*, *S4*, *S5*, *S6*, *S7*, *S8*, *S9*, *S10*, *S11*, *S12* and *S13*) of our EFSM. The following *@Action* annotated methods have been implemented: *insertCard()*, *validateCard()*, *requestPassword()*, *enterPassword()*, *validatePassword*, *requestValue()*, *enterValue()*, *validateBalance()*, *value()*, *unavailableBalance()*, *exit()* and *cardOut()*. In addition, the *getState* method, the *reset* method and context variables (*cardOk*, *pswOk* and *valueOk*) were defined.

The objects of type *User*, *ATM* and *Bank* that belong to the SUT were instantiated in the *AtmAdapter* class. In this class, a method was created for each event triggered in EFSM transitions. These methods (*insertCard()*, *validateCard()*, *enterPassword()*, *validatePassword()*, *enterValue()* and

¹<https://github.com/TESTSD2EFSM/SEKE2019>

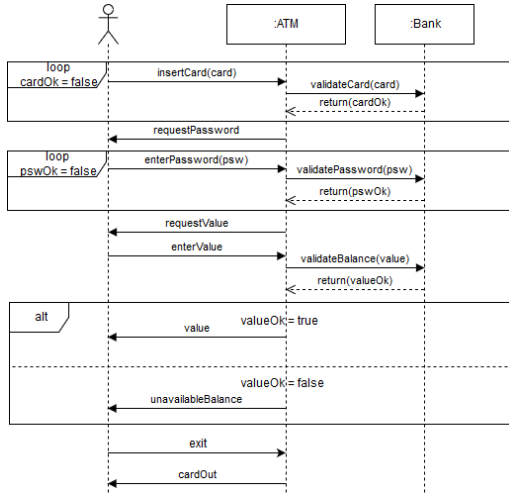


Fig. 4. ATM Sequence Diagram.

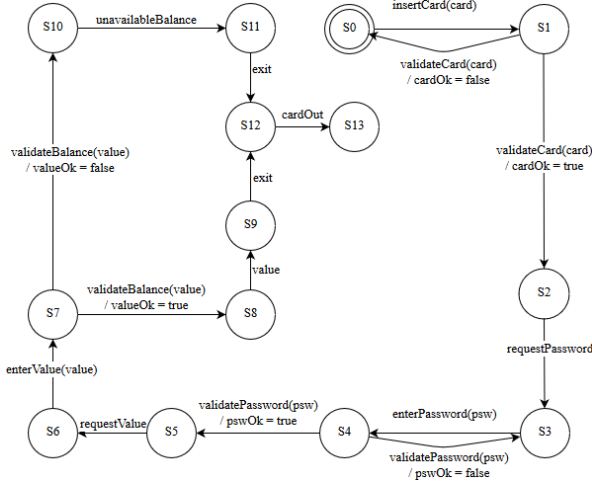


Fig. 5. ATM EFSM Model.

validateBalance()) are what make the communication of the model with the SUT.

In *ATMTest* class was instantiated the objects of type *ATMModel* and we used the *LookaheadTester* test strategy. To traverse all the transitions we configure the algorithm to generate a sequence of 70 test steps. To perform the tests we set the *card* attribute equal to 111, the *psw* attribute equal to 123 and the *balance* attribute equal to 100.00. These attributes belong to the Bank class of the SUT.

To verify the behavior of our approach, we performed the following test cases shown on the Table I. In addition to the test data (*card*, *psw* and *value*), the Table I shows action, state and transition coverages.

In addition to the metrics presented in Table I, the test cases *TestValidateCard01*, *TestValidateCard02*, *TestValidatePassword01*, *TestValidatePassword02*, *TestValidateBalance01* and *TestValidateBalance02* were concretized in Java through the JUnit library. Figure 6 shows an example of the

TABLE I
TEST CASES GENERATED.

id	card	psw	value	Action	State	Trans.
1	222	123	50	12/12	2/2	24/24
2	222	246	200	12/12	2/2	24/24
3	111	246	50	12/12	5/5	60/60
4	111	246	200	12/12	5/5	60/60
5	111	123	50	12/12	12/12	144/144
6	111	123	200	12/12	12/12	144/144

```

9  @Test
10 public void TestValidateCard01() {
11     Bank bank = new Bank();
12     boolean output = bank.validateCard("111");
13     assertTrue(output);
14 }

```

Fig. 6. Example of test case concretized in JUnit.

concrete test case of the *AtmJUnit* class.

These Java classes (*AtmModel*, *AtmAdapter*, *AtmTest* and *AtmJUnit*) are available in the *ModelJUnit/src/test/java* folder of the approach repository 1.

V. RELATED WORKS

One of the strengths of our approach is the automatic model transformation. As we have developed a tool to support our method, this task can be facilitated by the use of MDE concepts. Another advantage is the formulation of a UML model into formal model, since the UML has semantics problems and the formal models provide a set of techniques based on precise notation that can accurately translate the behavior of a system. In addition, since the main objective of our work is the generation of tests, our approach uses the ModelJUnit library to concretize the test cases in the Java programming language. On the other hand, we identified as a limitation of our work the use of only one UML diagram. Therefore, in this section of related works, we will compare our approach taking into consideration four aspects: used UML diagrams, tool support, use of formal models and concretization of test cases in some programming language.

In [18] is described a systematic test case generation method performed on Model-Based Testing (MBT) approaches by using UML Sequence Diagram. The UML Sequence Diagram is converted into a graph sequence and the graph is traversed to select the predicate functions. These predicates are transformed into Extended Finite State Machine (EFSM). From the EFSM, test cases are generated taking into account state coverage, transition coverage and action coverage. This technique is similar to ours, but EFSM is not automatically generated from the sequence diagram. Moreover, the technique does not use some important constructions of the Sequence Diagrams, such as the combined fragment. In this approach the test cases are concretized in the Java programming language.

In [19] an approach is presented to generate test cases using UML Activity and Sequence Diagrams. The approach consists of transforming the Sequence Diagram into a graph

called *Sequence Graph* and transforming the Activity Diagram into the *Activity Graph*. The software graph is formed by integrating the two graphs that are traversed to generate the test suite. The proposal uses UML models for generating tests, but differs from ours since it does not use MDE concepts and does not use formal models for test generation. In addition, the approach does not concretize test cases in some programming language.

In [20] an approach is presented to generate test cases using UML Sequence Diagrams. The approach consists of transforming Sequence Diagram in to Sequence Diagram Graph (SDG) and generate test cases from SDG. The Sequence Diagram is built with Object Constraint Language (OCL) and the SDG defines the activities as nodes and the interactions in the form of paths. The test case is generated by visiting the nodes and edges in the SDG. This proposal uses UML models to generate tests, but differs from ours since it does not use MDE concepts and formal models. In addition, test cases are not implemented in any programming language.

In the work of Seo et al. [16] is presented a method for generating test cases from Sequence Diagrams. This method suggests to generate test cases after conducting an intermediate transformation from a Sequence Diagram to an Activity Diagram. The proposal is similar to ours, since it uses model transformation, but does not use a formal model for generating test cases. Also we can not identify in the work if the transformation of models is carried out using MDE concepts, because it does not describe the manipulated metamodels in the process. In addition, the approach does not concretize test cases in some programming language.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we present a systematic procedure to generate test cases from UML Sequence Diagrams. Our approach uses concepts of Model-Driven Engineering to formalize UML Sequence Diagrams into EFSM and uses the ModelJUnit and JUnit libraries for automatic generation of test cases.

In Step 1, for the transformation of UML Sequence Diagram to EFSM, we perform the mapping of the elements of the respective metamodels through transformation rules. With this, we can provide a precise semantics to a widely used UML model.

In Step 2 of the approach, the formal model can be used as basis for automating the testing process, making it more efficient and effective. We used the ModelJUnit library to provide an interface to implement a formal test model, an adapter that communicates our model with the SUT and some test strategies already implemented. In addition, the execution of the tests is measured by coverage of state, actions and transitions. We use the JUnit library to perform tests in the Java programming language.

From the example, we can observe the applicability of our proposal, mainly in the generation of functional tests, since the approach starts with UML Sequence Diagrams that are important tools to model software scenarios and we end with test cases materialized in the Java programming language. These

tests were performed and metrics were generated allowing to analyze the behavior of the SUT according to the test model created.

As future work, other UML diagrams can be incorporated into the systematic procedure of generating tests and applying it to real examples through case studies or controlled experiments. In addition, the generated EFSM can be used for formal verification, such as checking safety, liveness and fairness properties.

REFERENCES

- [1] O. M. G. OMG. (2015) Unified modeling language 2.5. [Online]. Available: <http://www.omg.org/spec/UML/2.5/>
- [2] M. Petre, "Uml in practice," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 722–731.
- [3] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Softw. Test. Verif. Reliab.*, vol. 22, no. 5, pp. 297–312, 2012.
- [4] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, "Using formal specifications to support testing," *ACM Comput. Surv.*, vol. 41, no. 2, pp. 9:1–9:76, Feb. 2009.
- [5] ModelJUnit. (2010) The model-based testing tool. [Online]. Available: <https://sourceforge.net/projects/modeljunit/>
- [6] J. Bézivin, F. Jouault, and D. Touzet, "An introduction to the atlas model management architecture," 03 2005.
- [7] E. M. Framework. (2018) Acceleo. [Online]. Available: <https://www.eclipse.org/acceleo/>
- [8] S. Kent, "Model driven engineering," in *International Conference on Integrated Formal Methods*. Springer, 2002, pp. 286–298.
- [9] O. M. G. OMG. (2016) Mof - meta object facility. [Online]. Available: <http://www.omg.org/spec/MOF/>
- [10] R. Yang, Z. Chen, Z. Zhang, and B. Xu, "Efsm-based test case generation: Sequence, data, and oracle," *International Journal of Software Engineering and Knowledge Engineering*, vol. 25, no. 04, pp. 633–667, 2015.
- [11] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.
- [12] S. Sen, N. Moha, B. Baudry, and J.-M. Jézéquel, "Meta-model pruning," in *Model Driven Engineering Languages and Systems*, A. Schürr and B. Selic, Eds. Springer Berlin Heidelberg, 2009, pp. 32–46.
- [13] F. Fondement, P.-A. Muller, L. Thiry, B. Wittmann, and G. Forestier, "Big metamodels are evil," in *Model-Driven Engineering Languages and Systems*, A. Moreira, B. Schätz, J. Gray, A. Vallecillo, and P. Clarke, Eds. Springer Berlin Heidelberg, 2013, pp. 138–153.
- [14] R. Grønmo and B. Møller-Pedersen, "From sequence diagrams to state machines by graph transformation," in *Theory and Practice of Model Transformations*, L. Tratt and M. Gogolla, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 93–107.
- [15] Z. Micskei and H. Waeselynck, "The many meanings of uml 2 sequence diagrams: A survey," vol. 10, pp. 489–514, 10 2010.
- [16] Y. Seo, E. Y. Cheon, J. A. Kim, and H. S. Kim, "Techniques to generate utp-based test cases from sequence diagrams using m2m (model-to-model) transformation," in *IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, June 2016, pp. 1–6.
- [17] M. Utting, "How to design extended finite state machine test models in java," in *Model-Based Testing for Embedded Systems*, J. Zander, I. Schieferdecker, and P. J. Mosterman, Eds. Boca Raton, FL: CRC Press/Taylor and Francis Group, 2012, pp. 147–170.
- [18] V. Panthi and D. P. Mohapatra, "Automatic test case generation using sequence diagram," in *Proceedings of International Conference on Advances in Computing*, A. Kumar M., S. R., and T. V. S. Kumar, Eds. New Delhi: Springer India, 2012, pp. 277–284.
- [19] A. Tripathy and A. Mitra, "Test case generation using activity diagram and sequence diagram," in *Proceedings of International Conference on Advances in Computing*, A. Kumar M., S. R., and T. V. S. Kumar, Eds. New Delhi: Springer India, 2013, pp. 121–129.
- [20] M. MD* and B. GB, "A new approach to derive test cases from sequence diagram," *Information Technology & Software Engineering*, 2014.

The Smell of Blood: Evaluating Anemia and Bloodshot Symptoms in Web Applications

Zijie HUANG, Junhua CHEN, Jianhua GAO*

Department of Computer Science and Technology, Shanghai Normal University, Shanghai, 200234, China
hzjdev@foxmail.com, {chenjh, jhgao}@shnu.edu.cn

Abstract—In web applications that adopt layered architecture, Domain Layer is formed by Domain Model. Without any behavior, Anemic Domain Models contain only data. Those behaviors are dispersed into other layers and causing bloodshot symptoms in them. Most empirical studies suggest that symptoms of anemia and bloodshot degrade the maintainability of web applications, but no quantitative research has been done. This paper evaluates intensities of anemia and bloodshot symptoms based on metrics of three Code Smells, i.e. Data Class, Feature Envy and Blob. Furthermore, correlations of the intensities are evaluated using Spearman's rank correlation coefficient. The achieved results of experiments made on multiple versions of open-sourced projects show that over 65% of the applications are affected by anemia and bloodshot symptoms, and their intensities rarely decrease over time. Correlations of the intensities are also discovered within a single version and among multiple versions.

Keywords—anemic domain model; code smell; web application; domain driven design

I. INTRODUCTION

Domain Driven Design (DDD) [1] is a model-driven methodology aims to tackle the complexity of software systems. DDD introduced a layered architecture consisting of four layers including Interface, Application (also known as Service [2]), Domain, and Infrastructure Layer. Data in Domain Layer is presented by Domain Models. Fowler [3] defines the Domain Model as an object model of the domain that incorporates both behavior and data, while Anemic Domain Model (ADM) is a Domain Model containing little or no such behavior.

Applying ADMs to Domain Layer triggers the anemia of Domain Layer accompanied by the bloodshot of other layers. Firstly, the domain behaviors are dispersed into other layers notably the Service Layer, but those behaviors still depend on ADMs' data structure, causing tight coupling of the Service Layer to the Domain Layer. As a result, the Service Layer becomes oversized while its cohesion is reduced. Secondly, the object-oriented program degrades into process-oriented program [2] with reduced comprehensibility.

Evans [1] suggests the Service Layer should be kept "thin," while Fowler [2] concludes that ADM is a common anti-pattern and their usage should be avoided. Both of them point out that the core business logic of web applications should be concentrated on the Domain Layer. However, ADMs are still widely adopted in enterprise systems [4]. There have been several discussions questioning whether ADM is an anti-pattern, which suggests the advantages of ADM should be refocused, and in some cases, ADM may be the best practice [5, 6].

* Corresponding Author. The work of this paper was supported by the National Natural Science Foundation of China (Grants 61672355).
DOI reference number: 10.18293/SEKE2019-061

Above-mentioned symptoms and discussions are presented in empirical studies. To the best of our knowledge, the pros and cons of ADMs have not been quantified by any research.

Code Smell is the symptom of poor design and bad implementation choices [7]. Fowler [8] proposes 22 Code Smells for object-oriented programming including God Class (also known as Blob), Feature Envy, and Data Class. Code Smell intensities could be evaluated by proper metrics.

In this paper, we apply metrics of Blob and Feature Envy to quantify bloodshot symptoms, and Data Class for anemia symptoms. The source code of 112 MVC-based Java application together with 96 versions of 10 Java web application are analyzed, while over 65% of them are affected by anemia and bloodshot symptoms. The analysis shows that there is a positive correlation between anemia and bloodshot symptoms, and intensities of these two symptoms rarely decrease over time. The results also reveal that although ADMs are built to separate business logic and data structure completely, most ADM-based applications are not strictly following the design.

The main contributions of this paper are:

- 1) *Fills the gap in the quantification method of anemia and bloodshot symptoms in web applications.*
- 2) *Confirms Fowler's empirical discoveries of the negative impact of ADMs on software systems, while the advantages of ADMs have not been proved by any experiment.*
- 3) *Reveals the persistence and correlations of anemia and bloodshot symptoms.*

The rest of this paper is organized as follows. Section II introduces the background and related works. Section III details the Code Smell detection approach and the quantification methods of anemia and bloodshot symptoms. In Section IV we have done several experiments to verify the accuracy of our approaches and presented our evaluation process together with results. Then, we detail the threats that could affect the validity in Section V. Finally, Section VI concludes the paper.

II. RELATED WORKS

A. Code Smell Detection

This paper evaluates intensities of 3 following Code Smells.

- Blob is for an oversized class with low cohesion, and it implements multiple irrelevant responsibilities [9-11].
- Data Class is a class that contains only data but no behaviors [12].
- Feature Envy describes a method more interested in a class other than the one it is in [9].

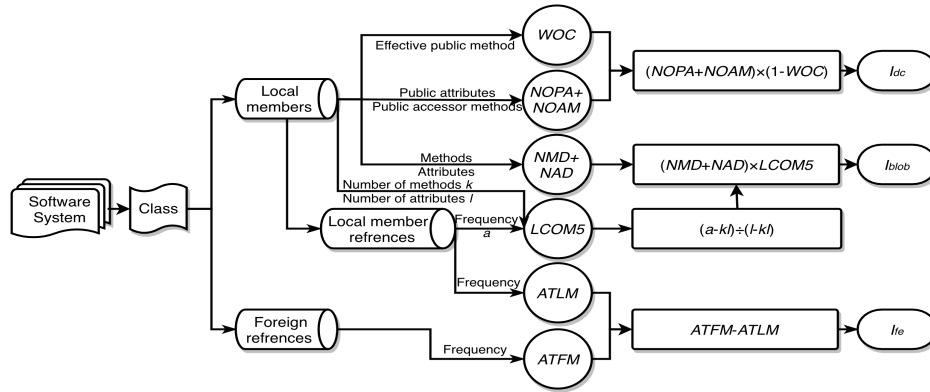


Fig. 1. Overview of Code Smell quantifying method

TABLE I. COMPARISON OF APPROACHES FOR FEATURE ENVY

Name / Approach	Lanza <i>et al.</i> [10]	Fokaefs <i>et al.</i> [14]
Couples With	Multiple classes	A single class
Precise Coupling Target	No. It detects coupling of a class generally.	Yes. It focus on both coupling source and target.
Metrics	Access to Foreign Data (ATFD), Local Attribute Access(LAA), Foreign Data Provider(FDP)	Access to Distinct Foreign Members, Access to Distinct Local Members
Chosen	No	Yes (Extended to detect multiple coupling targets)

TABLE II. COMPARISON OF APPROACHES FOR BLOB

Name / Approach	Lanza <i>et al.</i> [10]	Moha <i>et al.</i> [15]
Metrics	Structural: Access to Foreign Data (ATFD), Weighted Method Count (WMC), Tight Class Cohesion (TCC)	Structural: Number of Methods Defined (NMD), Number of Attributes Defined(NAD), Lack of Cohesion of Methods(LCOM5). Textual: Class name contains Manager, Process, Control, etc. Coupling: Access to at least one Data Class.
Chosen	No	Yes

Lanza *et al.* [10] quantified Code Smells and proposed several metrics and relevant thresholds, which are widely adopted in modern Code Analysis tools such as PMD [13]. JDeodorant [14] and DECOR [15] are notable tools detecting coupling and cohesion Code Smells.

Palomba *et al.* [11] proposed a pure textual detection approach called TACO, which is significantly different to traditional structural approach, aiming to discover conceptual coupling and cohesion problems. Furthermore, Palomba *et al.* [12] built a detection model based on multiple metrics, and they also evaluated the co-occurrence of Code Smells.

While Data Class metric is commonly accepted [10,12], multiple approaches of Feature Envy and Blob detection exists and their compatibilities to tasks in this paper is worth discussing. The difference in detection approaches is listed in Table I and Table II.

For Feature Envy, the main difference of the two approaches is whether an actual coupling target should be detected. Clarifying actual coupling targets is a must, as the identification of layer connections is vital. For Blob, both two methods refer to low cohesion, while Moha *et al.* [15]'s approach is more convenient due to its coupling detection and textual rules. In layered web application, classes and package names follow specific rules, and coupling with a Domain Model is the cause of the bloodshot symptom.

B. Web Application Design Problems

Aniche *et al.* [16] defined several MVC specific Code Smells, and evaluated their variation together with lifecycles based on a public dataset of 120 open-sourced GitHub repositories. This work developed metrics concerning specific web application code components such as Data Access Object (DAO), Repository, Controller and Service.

Cemus *et al.* and Cerny *et al.* [17,18] empirically investigated the negative impacts of ADMs and RDMs, and proposed a generic modeling method to ensure maintainability. According to their case studies, ADMs could cause Information Restatement and Concerns Tangling. RDMs also trigger coupling and cohesion problems, but intensities of design problems among Domains are decreased. The coupling problems within a single Domain could be resolved using Aspect Domain Model (AsDM).

C. Class Role Inference and Layering

Sakar *et al.* [19] generated Dependency Graph of modules and determined layer of each vertex according to their number of indegree and outdegree, while Hayashi *et al.* [20] inferred the role of code component in MVC-based applications using Dependency Graph.

Hickey *et al.* [21] split classes into layers according to their names, this method could lose its accuracy due to different naming strategies. Fokaefs [14] *et al.* and Aniche *et al.* [16] mentioned role detection of code components using class name and annotation name in their works.

III. APPROACH

Several fundamental data, i.e. Code Smell intensities and layers of classes, should be collected before evaluating anemia and bloodshot intensities. The overview of Code Smell quantifying method is illustrated in Fig. 1. In the following paragraphs, we explain each of the steps in detail.

A. Evaluating Data Class Intensity

Data Class is a class with interfaces that (i) provide almost no functionality and (ii) declare data fields. [10]

As shown in Fig. 2, *WMC* metric is used for (i), and *NOPA+NOAM* together with *WOC* are adopted for (ii).

WOC is the number of public methods (with accessors and constructors excluded) divided by the total number of public members. *NOPA* is the number of public attributes of a class, while *NOAM* is the number of accessor methods, i.e. getter and setter. *WMC* sums the complexity of all methods of a class.

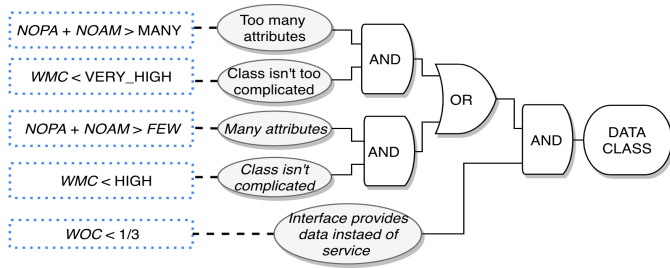


Fig. 2. Data Class detection approach

The *CYCLO* metric is used to calculate method complexity. The calculation approach of *CYCLO* defined in PMD following the standard rules given below is used in this paper:

- Methods have a base complexity of 1;
- +1 for every control flow statement (if, case, catch, throw, do, while, for, break, continue) and conditional expression (?);
- else, finally and default do not count;
- +1 for every Boolean operator (&&, ||).

Thus, the intensity of Data Class could be calculated as (1):

$$I_{dc}(C) = (1 - WOC(C)) \times (NOPA(C) + NOAM(C)) \quad (1)$$

B. Evaluating Feature Envy Intensities

A class is affected by Feature Envy if it access members of another class more frequently than its local members. [14]

Given a class $C_{current}$, the approach calculates *ATLM* as the frequency of distinct local member accessed by $C_{current}$. Then, a set C for all classes in the software system is formed, for each class C_i in C , the frequency of distinct member access from $C_{current}$ to C_i named a_i is evaluated. Finally, the classes in C is sorted by a_i in descending order. A class is affected by Feature Envy if the first class C_{top} is not equivalent to $C_{current}$.

For each a_i , calculate $diff = a_i - ATLM(C)$ and treat all negative $diff$ value as zero. Then we sum all $diffs$ to obtain the result of *ATFM* metric. For each $diff > 0$, the approach treats the related C_i as the coupling target of $C_{current}$.

The intensity is calculated as (2):

$$I_{fe}(C) = ATFM(C) - ATLM(C) \quad (2)$$

C. Evaluating Blob Intensities

As shown in Fig. 3, a Blob class is oversized, with low cohesion, having controller name pattern and couples with Data Classes. [15] We ignore name pattern as it is mentioned in Section D.

Size of a class could be measured according to the sum of *NMD* and *NAD* metrics. The cohesion of class could be evaluated by *LCOM5* metric, the main idea of *LCOM5* is to calculate the rate of access to local members.

Given a class C , the approach determines the number of method members k , the number of attribute members l , and the frequency of access to distinct local members a . *LCOM5* could be calculated as (3):

$$LCOM5(C) = \frac{a - kl}{l - kl} \quad (3)$$

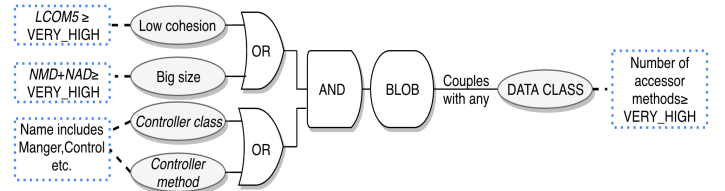


Fig. 3. Blob detection approach

TABLE III. CLASS ROLE INFERENCE APPROACH

Role/ Approach	Domain	Persistence	Service	Interface
Lowercase ed name includes	{domain, vo, entity, entities}	{dao, repo, repository}	service	{controller, ctrl, api}
Expected Layer	1 (bottom)	2	3	4 (top)

We pick the 3rd-quartile in Al Dallal's work [22] as threshold of *LCOM5* metric, and the fixed value in Palomba *et al.*'s [12] work as threshold of *NMD+NAD* metric.

The intensity of Blob could be calculated as (4):

$$I_{blob}(C) = (NMD(C) + NAD(C)) \times LCOM5(C) \quad (4)$$

D. Class Layering

There is no common and generic approach for class layering. Related works mainly consider two features, class dependencies [19,20] and class names [14,16,21]. This paper proposed a mixed layering approach that fits web applications.

First, the approach generates a Directed Acyclic Graph (DAG) according to class dependencies. Given a set of all classes in an application named C , a vertex is generated for each class. Then, pairs of vertexes are connected as follows: According to Table III, the likely role of each class could be inferred. For $C_i \in C$, $C_j \in C$ and $C_i \neq C_j$, if C_i accessed or called any member of C_j , and C_i and C_j have different inferred roles, an edge from C_i to C_j should be generated.

Then, the DAG should be split into 4 layers. At the very beginning, the approach splits the DAG into 3 layers. Vertexes with 0 in-degree are moved to the bottom (i.e. Domain) while those with 0 out-degree are moved to the top (i.e. Interface). For the rest of the vertexes kept in a temporary layer, we split them into 2 new layers using the similar method but ignore the connection of middle layer vertexes with the ones in the top and the bottom layer. Vertexes with 0 in-degree are moved to the lower layer (i.e. Persistence classes of Infrastructure), while other vertexes remain in the upper layer (i.e. Service).

Additionally, there exist some exceptions. For the vertexes whose roles cannot be inferred through names, their roles could be determined according to their actual layers as mentioned in Table III. If any Data Class has the naming pattern of Data Transfer Object and are only accessed by Interface Layer classes, they should be excluded from the DAG as their sole function is to normalize data during the transfer process.

Domains of an application could also be detected according to DAG and class name patterns. A set of words could be derived through splitting the Camel-cased Domain class names. If any word in the set appears in the name set of classes in other layers,

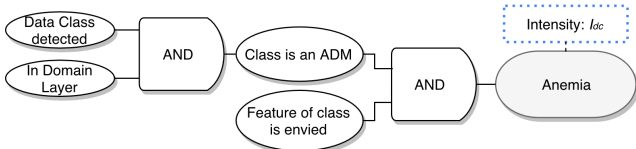


Fig. 4. The strategy of Anemic Class detection

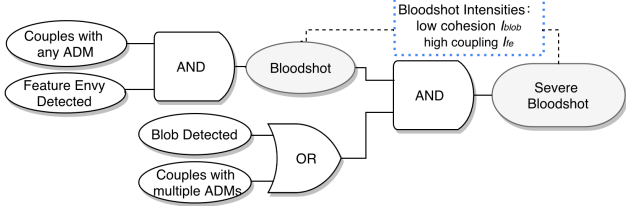


Fig. 5. The strategy of Bloodshot Class detection

and the two vertices of classes are connected, a domain could be determined.

E. Quantifying the symptoms of anemia and bloodshot

For now, we have collected the fundamental data including the layer of each class and their Code Smell intensities. The intensities of symptoms should be evaluated as follows.

As mentioned in Fig. 4, ADM is determined if a Data Class belongs to the Domain Layer, the intensity of anemia is I_{dc} .

As Fig. 5 illustrates, if a non-Domain-Layer class affected by Feature Envy and couples with a Data Class, we consider it bloodshot with intensities of two metrics: the extent of low cohesion I_{blob} and the extent of high coupling I_{fe} . For any bloodshot class with $I_{blob} > 0$ or couples with multiple ADMs, we regard it as a *Severe Bloodshot Class (SBC)*.

IV. EXPERIMENTS

This paper implements Code Smell metrics based on PMD [13] with necessary modifications [23]. Statistical data and plots are produced by Python scripts after generating reports of Code Smell intensities. Experiments are conducted to address the following 5 research questions:

- **RQ1 Accuracy:** *Is the approach able to evaluate class layers and Code Smell intensities accurately?*
- **RQ2 Severity:** *How severe is the symptom of anemia and bloodshot in web application?*
- **RQ3 Correlation:** *What is the relationship between the symptom of anemia and bloodshot?*
- **RQ4 Survivability:** *Do intensities of anemia and bloodshot symptom decrease over time?*
- **RQ5 Evaluation:** *What is the effect of applying ADM?*

A. Accuracy of fundamental data

A typical web application called military-shop is picked from the dataset [16] consisting of 120 open-sourced Java applications based on MVC architecture from GitHub. RQ1 is to be answered in this section using military-shop as a demo.

Precision and *Recall* metrics are used to evaluate accuracy. The metrics can be calculated as (5) and (6):

$$Precision = \frac{Correct \cap Detected}{Correct} \quad (5)$$

$$Recall = \frac{Correct \cap Detected}{Detected} \quad (6)$$

TABLE IV. ACCURACY OF LAYERING APPROACH

Role/Metric	Domain Model	Persistence	Service	Interface	Utility
Precision	100%	100%	100%	90%	85.71%
Recall	88.57%	100%	85.71%	94.73%	100%
Samples	35	7	14	19	12

TABLE V. LEVELS OF CORRELATION

Range of ρ	Correlation Level
[0.8,1.0]	Very Strong
[0.6,0.8)	Strong
[0.4,0.6)	Moderate
[0.2,0.4)	Weak
[0.0,0.2)	Very Weak

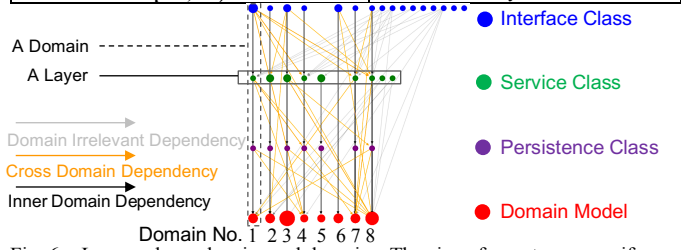


Fig. 6. Layers, dependencies and domains. The size of a vertex grows if overlapped. Edges are hidden if they connect vertices in the same layer.

where *Correct* denotes the set of items manually identified and *Detected* represents the set of items detected by the heuristic.

Fig. 6 shows the layered DAG of the project, while Table IV lists the accuracy of the layering approach. A few classes with ambiguous patterns were not correctly layered. The approach also detected all 8 domains of the project, and classes with correct inferred role were all placed into proper domain. The domain detection heuristic had a *Precision* of 100% and a *Recall* of 91.58%. Relevant Code Smell intensities were also validated.

Compared with the results of manual detection, we can conclude that fundamental data could be detected correctly. Manual detection is done independently by the first author and a developer having 3 years of enterprise web application development experience. A few disagreements were discussed and resolved later.

B. The Experiment conducted on single application

This section use Shopizer [24] as an example to demonstrate the experiment conducted on each project. The results will not be fully presented in this section as they are listed in Table VII and VIII instead.

For Q2, as shown in Table VII, 60.97% of the classes in the Domain Layer were ADMs, while Bloodshot occurred in 95.16% of the Interface and Service classes within detected Domain. About 70% bloodshot classes were SBCs.

In order to answer Q3, the correlations of I_{blob} , I_{fe} and I_{dc} should be evaluated. This paper uses the Spearman's rank correlation coefficient [25] with a *P*-value of 0.05 to analyze the correlation between every pair of intensities. The metric takes the input value of two sets of values equal in length and produces the correlation coefficient ρ together with the significance level *P*. With $P < 0.05$, the level of correlation could be considered statically significant with a level presented by ρ ranges in $[-1, 1]$ as shown in Table V. For example, the metric reported a strong

TABLE VI. DETECTION RESULTS OF 59 OPEN-SOURCED PROJECTS

	Class/ Domain Count	SBC rate for bloodshot classes		Bloodshot rate for Services and Inter faces of Domains	ADM contrib ution to Dom ain coupling	ADM rate for Domain Layer Classes	ρ of I_{blob} , I_{fe}	ρ of I_{dc} , I_{blob}	ρ of I_{dc} , I_{fe}
		Service	Interface						
Mean	645.34/23	65.30%	65.98%	97.76%	49.88%	50.64%	0.68	0.07	0.09
Variance	333845.04/330	0.09	0.09	0.01	0.05	0.05	0.02	0.15	0.11

TABLE VII. PROJECT COMPOSITIONS OF 10 OPEN-SOURCED JAVA WEB APPLICATIONS

Project Name	Commit/Fork/ Release	Latest Version	Class/Do main Count	SBC rate for bloodshot classes		Bloodshot rate for Services and Inter faces of Domains	ADM contrib ution to Dom ain coupling	ADM rate for D omain Layer Cl asses
				Service	Interface			
Shopizer	193/941/6	2.2.0	811/31	72.22%	68.18%	95.16%	68.85%	60.97%
OpenLegislation	3192/88/28	2.17	787/36	95.65%	61.11%	98.96%	32.70%	32.04%
LibrePlan	9657/148/32	1.4.1	1294/50	66.67%	100%	99.14%	41.33%	13.91%
OpenCMS	22750/321/228	10.5.4	3382/54	84.61%	76.92%	97.67%	43.63%	18.84%
Thingsboard	1514/676/17	2.1	797/31	100%	63.64%	98.73%	35.67%	8.70%
Sakai	46898/535/21	12.3	4951/45	71.43%	92.00%	100%	38.85%	30.86%
OpenClinica	8521/167/30	4.5.2	1436/38	81.48%	65.06%	96.85%	71.94%	58.41%
Apollo	1944/2873/14	1.0.0	458/24	68.32%	78.27%	100%	57.79%	78.92%
Dataverse	12042/180/30	4.9.2	675/19	33.33%	55.56%	97.43%	44.30%	42.30%
DDDLib	2310/153/18	4.6.1	392/-	0%	0%	-	0%	10.00%

TABLE VIII. P VALUES OF 10 OPEN-SOURCED JAVA WEB APPLICATIONS

Project Name	Analyzed Versions	ρ of I_{blob} , I_{fe}	ρ of I_{dc} , I_{blob}	ρ of I_{dc} , I_{fe}	ρ of ΔI_{blob} , ΔI_{fe}	ρ of ΔI_{dc} , ΔI_{blob}	ρ of ΔI_{dc} , ΔI_{fe}	R_{dec}
Shopizer	6	1.00	-	-	0.78	0.70	0.79	0.70%
OpenLegislation	10	0.81	0.71	0.94	0.67	-	0.31	0.00%
LibrePlan	10	-	-	0.85	-	0.73	0.64	3.84%
OpenCMS	10	0.80	-	-	0.95	0.58	0.47	1.72%
Thingsboard	10	0.99	0.62	-	0.81	0.61	0.74	2.38%
Sakai	10	0.94	-	0.63	0.87	0.51	0.38	1.47%
OpenClinica	10	0.57	-	0.71	0.70	0.28	0.51	0.45%
Apollo	10	1.00	0.96	0.96	0.70	0.36	0.42	1.11%
Dataverse	10	-	-	-	0.78	-	-	0.00%
DDDLib	10	-	-	-	-	-	-	0.00%

correlation between I_{blob} and I_{fe} with $\rho=0.71$ ($P=2.62e-12$).

In order to answer Q4, it is necessary to analyze the variation of the 3 concerned intensities (ΔI_{blob} , ΔI_{fe} and ΔI_{dc}) among all 6 release versions. Intensities between two neighboring versions were calculated and normalized to the range [0,1]. The correlations of variations were also evaluated.

For Q5, we calculated the rate of ADMs' contributions to I_{fe} in domains, i.e. the sum of I_{fe} of every domain class coupled with at least 1 ADM divided by the sum of I_{fe} in all domains.

C. Results

Following a similar process of Section B, the experiment was conducted on 120 applications mentioned in Section A, in which 112 projects were available for access on GitHub. Among the 112 projects, 66.96% of them were affected by anemia and bloodshot symptoms (ADM rate > 0% and bloodshot class rate > 0%) and 52.68% of them had at least 1 valid ρ value ($P < 0.05$ and $0 < |\rho| < 1$). Table VI reports the result of the applications with valid ρ value.

The primary cause of the invalid ρ values were: (i) The project was too lightweight, i.e. contained few lines of code. (ii) No such correlation exists. Domains were not detected in some of the applications, in most of the cases they did not have valid ρ values owing to the fact that these projects were not layered or they were not web applications.

This dataset was collected using the filter "exists at least 10 controllers" for the analysis of MVC-based applications, most of them lacked valid release information, which were not capable for analysis based on multiple versions. So a new dataset must be picked.

We selected 10 Java web applications with more than 100 commits, more than 100 classes, and at least one commit in recent 6 months. For each project, we analyzed its latest 10 versions available. 9 out of the 10 projects were affected by anemia and bloodshot symptoms. DDDLlib was an exception that follows the DDD specification. Dataverse did not have a clear layering pattern, and the size of each layer was not enough for the metrics to evaluate correlation data.

"The proportion of decreasing anemia or bloodshot intensities" named R_{dec} was calculated as the ratio of "the number of classes with any of the three intensities decreased" to "the number of SBCs and ADMs".

Table VII reports the composition of the applications, while Table VIII lists multiple correlation coefficients (ρ), any ρ with $P > 0.05$ will be marked as unavailable(-).

D. Discussions

For Q2, more than 65% of the 112 projects analyzed by the experiment were affected by anemia and bloodshot symptoms. For web application domains based on ADMs, the proportion of bloodshot classes exceeded 90%, and most of them were SBCs, indicating the symptoms were common and severe.

For Q3, regarding the ρ values of Table VI and Table VIII, the two intensities of bloodshot correlated in most of the cases. Among different versions of the same project, the variations of the three intensities often correlated. We also analyzed the correlation of symptom intensities in different layers within single domains. But we did not find any significant correlation. The cause might be a large number of design problems within the domain are related to other domains instead of itself.

For Q4, as shown in the last column of Table VIII, the symptoms of anemia and bloodshot rarely reduced, which also confirms the conclusion about structural Code Smells that they tend to become more severe and are rarely removed [9,15].

For Q5, the result of our experiment is not showing any advantages of ADM, but confirmed the widely-accepted conclusion that there are a lot of coupling and cohesion problems within an ADM-based domain resulting in SBCs. To our astonishment, applying ADM will not result in a complete separation of data and business logic as it is designed for in most of the cases. The ADM-based applications often contain 30% to 70% of non-ADM domain models, in which domain behaviors are implemented. In conclusion, ADM has an obvious shortage of keeping single responsibilities.

V. THREATS TO VALIDITY

A threat to Internal Validity is that the layering approach uses name patterns. If the layering pattern in class name is ambiguous, the detection will be completed only according to dependency information, and accuracy will be affected.

Threats to External Validity are listed as follows: (1) Detection process could lose its validity on small applications, as thresholds derive from enterprise applications. (2) There exist a few applications that do not follow layered design. (3) Our approach analyses Java-based web application, the conclusion may not satisfy applications based on weakly-typed languages.

VI. CONCLUSIONS AND FUTURE WORK

It has been 15 years since Fowler first proposed the concept of ADM and its negative impacts, but ADM-based domain modeling is still popular. This paper analyzed source code of 112 MVC patterns based Java applications in a public dataset and 96 versions of 10 Java web applications, and concluded that over 65% of applications are affected by anemia and bloodshot symptoms. The analysis also suggests a positive correlation between the two symptoms, and they rarely decrease over time. The shortage of ADM are confirmed by experiment results, furthermore, in most of the cases, the complete separation of data and business logic are not implemented as ADMs are designed for.

Our future work involves the investigation of the impact of commit changes on anemia and bloodshot symptoms, and the application of Deep Learning approaches to improve the accuracy of class role detection is also worth trying.

REFERENCES

- [1] E. Evans, *Domain-driven design: tackling complexity in the heart of software*. Boston: Addison-Wesley Professional, 2004.
- [2] AnemicDomainModel[Online]. Available: <https://www.martinfowler.com/bliki/AnemicDomainModel.html>. [Accessed Feb 28, 2019].
- [3] M. Fowler, *Patterns of enterprise application architecture*. Boston: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [4] F. Wang, L. Yan, Z. Peng, S. Wei, and D. Yuan. "The investigation of WEB software system based on domain-driven design." in *International Conference on Web Information Systems and Mining*, Taiyuan, China, 2011, pp. 11-18.
- [5] The Anemic Domain Model is no Anti-Pattern, it's a SOLID design [Online]. Available: <https://blog.inf.ed.ac.uk/sapm/2014/02/04/the-anaemic-domain-model-is-no-anti-pattern-its-a-solid-design/> [Accessed Feb 28, 2019].
- [6] R. Wirfs-Brock. "Are software patterns simply a handy way to package design heuristics?." in *Proceedings of the 24th Conference on Pattern Languages of Programs*, Vancouver, Canada, 2017, p. 3.
- [7] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia *et al.*, "When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away)," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1063-1088, 1 Nov. 2017.
- [8] M. Fowler, K. Beck, J. Brant, W. Opdyke and D. Roberts. *Refactoring: improving the design of existing code*. Boston: Addison-Wesley Professional, 1999.
- [9] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto and A. De Lucia, "The Scent of a Smell: An Extensive Comparison Between Textual and Structural Smells," *IEEE Transactions on Software Engineering*, vol. 44, no. 10, pp. 977-1000, 1 Oct. 2018.
- [10] M. Lanza, R. Marinescu. *Object-oriented metrics in practice: Using software metrics to characterize, evaluate, and improve the design of object-oriented systems*, Berlin: Springer Science & Business Media, 2007
- [11] F. Palomba, A. Panichella, A. De Lucia, R. Oliveto and A. Zaidman, "A textual-based technique for Smell Detection," in *IEEE 24th International Conference on Program Comprehension*, Austin, TX, USA, 2016, pp. 1-10.
- [12] F. Palomba, M. Zanoni, F. A. Fontana, A. De Lucia and R. Oliveto, "Toward a Smell-Aware Bug Prediction Model," *IEEE Transactions on Software Engineering*, vol. 45, no. 2, pp. 194-218, 1 Feb. 2019.
- [13] PMD[Online]. Available: <https://pmd.github.io> [Accessed Feb 28, 2019].
- [14] M. Fokaefs, N. Tsantalis and A. Chatzigeorgiou, "JDeodorant: Identification and Removal of Feature Envy Bad Smells," in *IEEE International Conference on Software Maintenance*, Paris, France, 2007, pp. 519-520.
- [15] N. Moha, Y. Gueheneuc, L. Duchien and A. Le Meur, "DECOR: A Method for the Specification and Detection of Code and Design Smells," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 20-36, Jan.-Feb. 2010.
- [16] M. Aniche, G. Bavota, C. Treude, M. Gerosa, and A. Deursen, "Code smells for model-view-controller architectures," *Empirical Software Engineering*, vol. 23, no. 4, pp. 2121-2157, Aug. 2018.
- [17] K. Cemus, T. Cerny, L. Matl and J. Michael, "Aspect, Rich, and Anemic Domain Models in Enterprise Information Systems," in *International Conference on Current Trends in Theory and Practice of Informatics*, Harrachov, Czech, 2016, pp. 445-456.
- [18] T. Cerny, M. Donahoo, "How to reduce costs of business logic maintenance," in *IEEE International Conference on Computer Science and Automation Engineering*, Shanghai, China, pp. 77-82
- [19] S. Sarkar, G. M. Rama and S. R., "A Method for Detecting and Measuring Architectural Layering Violations in Source Code," in *13th Asia Pacific Software Engineering Conference*, Bangalore, India, 2006, pp. 165-172.
- [20] S. Hayashi, F. Minami, M. Saeki, "Detecting Architectural Violations Using Responsibility and Dependency Constraints of Components," *IEICE TRANSACTIONS on Information and Systems*, vol. 101, no. 7, pp. 1780-1789, 1 Jul. 2018.
- [21] S. Hickey, M.O. Cinnéide, "Search-Based Refactoring for Layered Architecture Repair: An Initial Investigation," in *Proceedings of the North American Search Based Software Engineering Symposium*, Dearborn, MI, USA, 2015, pp. 1-16.
- [22] J. Al Dallal, "Measuring the Discriminative Power of Object-Oriented Class Cohesion Metrics," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 788-804, Nov.-Dec. 2011.
- [23] Our fork of PMD[Online]. Available: <https://github.com/CodeSmellID/pmd-mini> [Accessed Feb 28, 2019].
- [24] Shopizer[Online]. Available: <https://github.com/shopizer-ecommerce/shopizer> [Accessed Feb 28, 2019].
- [25] J. H. Zar, "Significance testing of the Spearman rank correlation coefficient," *Journal of the American Statistical Association*, vol. 67, no. 339, pp. 578-580, 1 Oct. 1

Formalization and Verification of RTPS StatefulWriter Module Using CSP

Jiaqi Yin¹

Huibiao Zhu^{*1}

Yuan Fei^{*2}

Qiwen Xu³

Ruobiao Wu⁴

¹Shanghai Key Laboratory of Trustworthy Computing
East China Normal University, Shanghai, China

²School of Information, Mechanical and Electrical Engineering
Shanghai Normal University, Shanghai, China

³Faculty of Science and Technology, University of Macau, China

⁴Huawei Technology Co., Ltd. China

Abstract—The Real Time Publish Subscribe protocol (RTPS), as a Data Distribution Service (DDS) protocol for computer systems, is composed of several modules. We focus on RTPS StatefulWriter Module which has two patterns, reliable pattern and best-effort pattern. As the main module of sending and receiving messages, its security and reliability are of great concern. The formal method can analyze whether it is a highly credible model from the mathematical point of view. Our research pays attention to the reliable pattern. Thus it is of great importance to model and verify whether the pattern is reliable through formal methods. In this paper, we model seven components of the module using Communicating Sequential Processes (CSP). By feeding the models into the model checker Process Analysis Toolkit (PAT), we verify four properties, divergence free, acknowledgement mechanism, data consistency and sequentiality. Consequently, it can be apparently concluded that the pattern of this module is reliable, which totally caters for its specification.

Index Terms—RTPS StatefulWriter Module, CSP, PAT, Modeling, Verification

I. INTRODUCTION

Data Distribution Service (DDS) is a new generation of distributed real-time communication middleware technology specification developed by Object Management Organization (OMG) based on HLA and CORBA standards. It adopts publish/subscribe architecture, emphasizes data-centric and provides abundant quality of service strategies. The Real Time Publish Subscribe protocol (RTPS), as a Data Distribution Service (DDS) protocol for computer systems, transfers data from publishers to subscribers. StatefulWriter module is one module of RTPS protocol. It has two modes, which are reliable pattern and best-effort pattern. Reliable pattern means the data must be always transferred to subscribers in the specification. Thus, we follow with interest the reliability of the reliable pattern in the module.

The behavior of the module contains acknowledgement mechanism and heartbeat mechanism. The former guarantees all messages to be received by subscribers and the latter assures the messages to reach the subscribers. Besides, data consistency and sequentiality need to be ensured in the reliable pattern. Our work is to model and verify the reliable pattern of

the module. Thus, through formal modeling and verification of StatefulWriter module, the specification can be more precisely modeled and validated, avoiding the ambiguity of natural language description, which has certain guiding significance.

The most related prior work we identified is a study by Liu et al. [5] that mainly verified the security, activity and priority of DDS in ROS2. In addition, Alaerjan et al. [1] defined the missing functional behavior in DDS dynamic model and the semantics of the new operation using Object Constraint Language (OCL). Some recent research projects [2], [7], [10] have explored analysis and verification of many aspects of DDS, such as real-time performance, security of DDS-based middleware and so on. Our work focuses on the communication dependability of the module's reliable pattern using formal methods.

The remainder of this paper is organized as follows. Section II gives a brief introduction to RTPS StatefulWriter Module, the process algebra CSP and model checker PAT. In Section III, we formalize the seven core components in the module using CSP. We apply the model checker PAT to implement the model and verify four properties in Section IV, including divergence free, acknowledgement mechanism, data consistency and sequentiality. Section V describes the conclusion and future work.

II. BACKGROUND

This section detailedly describes the flows of the module which are used in the next section and briefly introduces the process algebra CSP and model checker PAT.

A. RTPS StatefulWriter Module

RTPS StatefulWriter Module has seven components. There are Publisher, DDSWriter, RTPSWriter, HistoryCache, Subscriber, DDSReader and RTPSReader. Fig. 1 shows the 22 communications in the module. It can be divided into four submodules, which are writing data, heartbeat mechanism, reading data and removing data. Here we combine all of them in Fig. 1. The detailed messages are as follows.

Writing data submodule contains the first six interactions. Publisher writes data by invoking the **write** operation on

*Corresponding authors: hbzhu@sei.ecnu.edu.cn (H. Zhu).
yuanfei@shnu.edu.cn (Y. Fei).

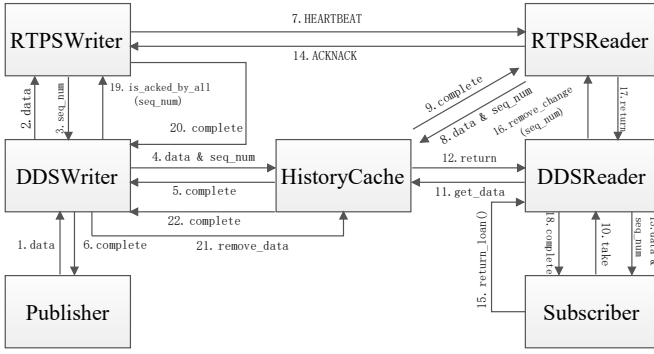


Fig. 1. Communications of RTPS StatefulWriter Module

DDSWriter. Then, DDSWriter invokes the **new_change** operation on RTPSWriter to create a new CacheChange. Each CacheChange has a unique sequence number. Also, DDSWriter uses the **add_change** operation to store the CacheChange into RTPSWriter's HistoryCache. When functions are invoked, they return the message that means operation has been executed successfully.

HeartBeat submodule is used to send message heartbeat to Reader endpoint. If the message is received smoothly within the specified time and checked by the Subscribers, RTPSWriter receives the information **ACKNACK** indicating confirmation.

Reading data submodule consists of four interactions. Subscriber reads data by invoking the **take** operation in DDSReader. Then, DDSReader accesses the changes with data and sequence number from HistoryCache. Ultimately, the **take** operation returns the data and sequence number to Subscriber.

Removing data submodule is composed of the remaining communications. Subscriber invokes the **return_loan** operation on DDSReader to notify that it no longer uses the data. Next, DDSReader uses the **remove_change** operation to remove the data from HistoryCache. Then, DDSWriter invokes the **is_acked_by_all** operation to determine whether all the changes are all received by the Reader endpoints. At length, DDSWriter calls the **remove_change** operation to remove the data from HistoryCache.

B. A Brief Introduction to CSP and PAT

CSP [3], [4] is a process algebra proposed by Hoare in 1978. As one of the most mature formal methods, it is tailored for describing the interaction between concurrent systems by mathematical theories. For its well-known expressive ability, CSP has been widely used in many fields [6], [8], [9]. CSP processes are constituted by primitive processes and actions. We use the following syntax to define the processes in this paper, whereby P and Q represent processes, a and b denote the atomic actions and c stands for the name of a channel.

$$P, Q = \text{Skip} \mid \text{Stop} \mid a \rightarrow P \mid c?x \rightarrow P \mid c!e \rightarrow P \mid \\ P \square Q \mid P \parallel Q \mid P \mid\mid Q \mid P;Q \mid P[[X]]Q$$

where:

- *Skip* stands for a process which only terminates successfully.
- *Stop* represents that the process does nothing and its state is deadlock.
- $a \rightarrow P$ first performs action a , then behaves like P .
- $c?x \rightarrow P$ receives a message by channel c and assigns it to variable x , then behaves like P .
- $c!e \rightarrow P$ sends a message e through channel c , then performs P .
- $P \square Q$ acts like either P or Q and the environment decides the selection.
- $P \parallel Q$ shows the parallel composition between P and Q . The \parallel means that actions in the alphabet of both operands require simultaneous participation of them.
- $P;Q$ executes P and Q sequentially.
- $P[[X]]Q$ indicates that processes P and Q perform the concurrent events on the set X of channels.

PAT Analysis Toolkit (PAT), is designed as an extensible and modularized framework for automatic system analysis based on CSP. It supports specifying and verifying systems in many different modeling languages and there are already various systems such as concurrent real-time systems, probabilistic systems, activity recognition and in other domains that have been verified in PAT. PAT can be applied in verifying various properties such as divergencefree, reachability and LTL propertites with assertions in distributed systems. Here we list some notations as below.

- $\#define N \ 0$ defines a global constant N with the initial value 0.
- *channel* $c \ 1$ stands for a channel which has the name c and the buffer size 1.
- $var \ cond = false$ represents a boolean condition with the initial value *false*.
- $[cond] \ P$ indicates a guarded process, which only executes when its guard condition is satisfied.
- $\#define \text{goal } n>0; \ \#assert \ P \text{ reaches goal};$ defines an assertion that checks whether process P can reach a state where the condition goal is satisfied.
- $\#assert \ P() \mid = F;$ defines an assertion that checks whether process P satisfies the formula F .

III. MODELING RTPS STATEFULWRITER MODULE

In this section, we give the formal model of RTPS StatefulWriter Module. The formalization is proceeded based on the communications in Fig. 1. Our model is constituted by seven core components: *Publisher*, *DDSWriter*, *RTPSWriter*, *HistoryCache*, *Subscriber*, *DDSReader* and *RTPSReader*.

A. Sets, Messages and Channels

Fig. 2 gives the channels of communication in the module. For more convenience, we give the definitions of sets used in

the model. We define the set of **Publisher** of Publisher component, **DDSWriter** of DDSWriter component, **RTPSWriter** of RTPSWriter component, **HistoryCache** of HistoryCache component, **DDSReader** of DDSReader component and **RTPSReader** of RTPSReader component. In addition, we define the set: **REQ** of request, **SEQ** of sequence number messages and **DATA** of data information; for simplicity, **ALLSETS** defines the unions of all sets of RTPS StatefulWriter module.

Based on the sets defined above, the messages transmitted among components are defined as follows:

$$\begin{aligned}
 MSG &= MSG_{req} \cup MSG_{rep} \cup MSG_{data} \\
 MSG_{req} &= \{msg_{req}.A.B.content \mid A \in (ALLSETS-Publisher), \\
 &\quad B \in ALLSETS, content \in REQ\} \\
 MSG_{rep} &= \{msg_{rep}.A.B.content \mid A \in ALLSETS, \\
 &\quad B \in ALLSETS, content \in SEQ \cup REQ\} \\
 MSG_{data} &= \{msg_{data}.A.B.content \mid A \in ALLSETS, \\
 &\quad B \in ALLSETS, content \in DATA\}
 \end{aligned}$$

where, MSG_{req} represents the set of request messages, MSG_{rep} stands for the set of all kinds of response requests and MSG_{data} represents the set of messages transmitting data. Each message contains a tag from the set $\{msg_{req}, msg_{rep}, msg_{data}\}$.

Then, we give the definitions of channels. In this paper, the channels using **COM_PATH** to represent can be defined as follows:

$$\begin{aligned}
 &ComPW, ComWP, ComWR, ComRW, ComWC, \\
 &ComCW, ComCT, ComTC, ComRT, ComTR, \\
 &ComCD, ComDC, ComSD, ComDS, ComDT, ComTD
 \end{aligned}$$

The declarations of the channels are as follows:

Channel $COM_PATH : MSG$

Table I shows the meanings and functionalities of representative messages transferred in the channels.

TABLE I
THE EXPLANATIONS OF TYPICAL MESSAGES OF THE MODEL

Messages	Functionalities
<i>data, DATA</i>	data transferred in the module
<i>seq_num, SEQ_NUM</i>	sequence number
<i>heartbeat</i>	judge whether data is received within required time
<i>noinvoke</i>	judge whether invoke functions
<i>complete</i>	judge whether execute the function
<i>take</i>	read data from cache
<i>remove</i>	remove data from cache
<i>get_change</i>	get changes from cache

B. Overall Modelling

System process is composed of all seven subprocesses running in parallel through their own corresponding channel. The subprocesses are *Publisher*, *DDSWriter*, *RTPSWriter*, *HistoryCache*, *Subscriber*, *DDSReader* and *RTPSReader*. The behavior of System process is modelled as below.

$$\begin{aligned}
 System &=_{df} Publisher \parallel DDSWriter \parallel RTPSWriter \parallel \\
 &\quad HistoryCache \parallel Subscriber \parallel DDSReader \parallel RTPSReader
 \end{aligned}$$

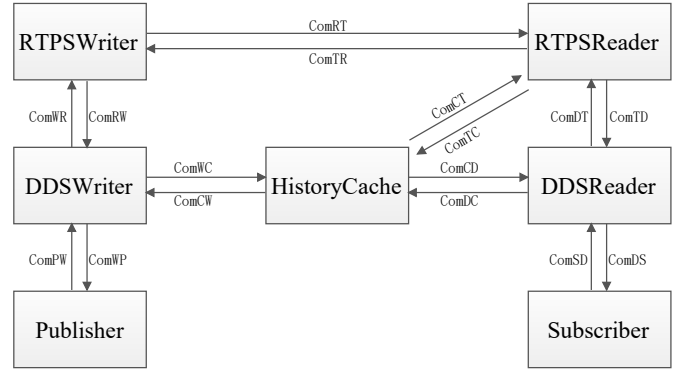


Fig. 2. Channels of RTPS StatefulWriter Module

C. Publisher

Publisher process is the core part in writing data submodule. It is used to write data to HistoryCache and receives the complete information from DDSWriter. The behavior of *Publisher* process is modelled as below.

$$\begin{aligned}
 Publisher() &=_{df} ComPW!msg_{data}.P.W.data \\
 &\quad \rightarrow ComWP?msg_{rep}.W.P.complete \rightarrow Publisher()
 \end{aligned}$$

D. DDSWriter

DDSWriter process plays an important role in writing data and removing data submodule. First, it sends and receives messages to write data from Publisher. Then, it applies the acknowledgement mechanism to check if the data has been totally received. If the return message is *ACK*, it invokes function to remove data and sequence numbers from HistoryCache. The behavior of *DDSWriter* process is modelled as below.

$$\begin{aligned}
 DDSWriter() &=_{df} ComPW!msg_{data}.P.W.data \\
 &\quad \rightarrow ComDR?msg_{req}.W.R.data \\
 &\quad \rightarrow GetSeqNum(); ComRD?msg_{rep}.R.D.complete \\
 &\quad \rightarrow ComWC!msg_{req}.W.C.data.seq_num \\
 &\quad \rightarrow ComCW?msg_{rep}.C.W.complete \\
 &\quad \rightarrow ComWP!msg_{rep}.W.P.complete \\
 &\quad \rightarrow if (id_acked_by_all(seq_num) == true) \{ \\
 &\quad \quad ComRD?msg_{rep}.R.D.complete \\
 &\quad \quad \rightarrow ComWC!msg_{req}.W.C.remove \\
 &\quad \quad \rightarrow remove_change(seq_num); \\
 &\quad \quad ComCW?msg_{rep}.C.W.complete \\
 &\quad \quad \rightarrow DDSWriter() \} else \{Skip\}
 \end{aligned}$$

In the above formula, *GetSeqNum()* is used to set the number of the sequence; *id_acked_by_all(seq_num)* is a function that judges whether the data with the *seq_num* is acknowledged; *remove_change(seq_num)* is used to remove the changes in the HistoryCache component.

E. RTPSWriter

RTPSWriter process works in writing data and heartbeat mechanism submodule. First, it produces the unique sequence number for the uploaded data. Second, it uses heartbeat mechanism to send heartbeat to RTPSReader for assuring the data can be transferred within the required interval. Finally, it helps to check whether the sequence numbers are checked by

all Subscribers. The behavior of *RTPSWriter* is modelled as below.

```

RTPSWriter() =df ComDR?msgreq.D.R.data
  → ComRD!msgrep.R.D.complete
  → DATAHeartBeat(); ComRW?msgrep.R.W.ACKNACK
  → if(head(ACKNACK)==ACK){
    acked_changes_set(seq_num); ComDR?msgreq.D.R.seq_num
    → ComRD!msgrep.R.D.complete
    → RTPSWriter() } else {Skip}

```

ACKNACK contains *ACK* and *seq_num*, so we use *head(ACKNACK)* to retrieve the message *ACK*. *acked_changes_set(seq_num)* checks whether the changes are set. HeartBeat mechanism is very important in the model. Its detailed behavior is modelled as follows:

```

DATAHeartBeat() = Clock(0)|{time}|SendHBeat();
Clock(i) = (tick → Clock(i+1))
  □(time?request → time!i → Clock(i));
SendHBeat() = time!request → time?startTime1{
  startTime=startTime1
  if (lastTime - startTime > HBeatInterval){
    SendHBeat()
  } else{ ComWR!msgreq.W.R.heartbeat
    → event{lastTime=startTime; }
    → SendHBeat() }

```

time is the channel between *Clock* and *SendHBeat()*; *Clock(i)* process returns the current time if receives the *request* message. *SendHBeat()* process sends the *heartbeat* message if the time difference is less than *HBeatInterval*; otherwise, the process cycle continues.

F. HistoryCache

HistoryCache process is like a database mainly for storing data and corresponding sequence number. It functions in every submodule, such as writing data and removing data. When receiving the request from Publishers or Subscribers, it invokes the homologous function to handle. The behavior of *HistoryCache* process is modelled as below.

```

HistoryCache() =df ComWC?msgreq.W.C.data.seq_num
  → ComCW!msgrep.C.W.complete
  → ComTC?msgreq.T.C.data.seq_num
  → ComCT!msgrep.C.T.complete
  → complete12:=get_changes(seq_num);
    ComCR!msgrep.C.R.complete12
  → ComWC?msgreq.W.C.remove_change
  → complete23:=remove_changes(seq_num);
    ComCW!msgrep.C.W.complete23
  → HistoryCache()

```

In the above formula, function *get_changes(seq_num)* and *remove_changes(seq_num)* is used to get and remove changes from *HistoryCache* component, respectively. Both of them can return the value 1 to indicate the operation is successful; otherwise, they return 0.

G. Subscriber

Subscriber process is designed for reading data and removing data submodule. First of all, it calls *take* function to receive data from *HistoryCache*. Next, it notifies other components that the data will not be used and gets the corresponding feedback. The behavior of *Subscriber* process is modelled as below.

```

Subscriber() =df ComSR!msgreq.S.R.take
  → DATA := take(); ComRS?msgrep.R.S.DATA
  → ComSR!msgreq.S.R.loan → noinvoke := return_loan();
    ComRS?msgrep.R.S.noinvoke → Subscriber()

```

In the above formula, function *take()* reads data from *HistoryCache* component and *return_loan()* indicates the data is not invoked any more, whose value is assigned to *noinvoke*.

H. DDSReader

DDSReader process is used for reading data and removing data submodule. First, it helps the *Subscriber* get data and sequence number from *HistoryCache*. Then, it invokes *remove_change* function to remove changes in *HistoryCache*. The behavior of *DDSReader* is modelled as below.

```

DDSReader() =df ComSR?msgreq.S.R.take
  → ComRC!msgreq.R.C.get_change → ComCR?msgrep.C.R.complete
  → ComRS!msgrep.R.S.DATA → ComSR?msgreq.S.R.loan
  → ComRS!msgrep.R.S.noinvoke → ComDT!msgreq.D.T.remove
  → noinvoke2 := remove_changes(); ComTD?msgrep.T.D.noinvoke2
  → DDSReader()

```

In the above formula, function *remove_changes()* is the same as that in process *HistoryCache*. *take* and *get_change* are the messages to invoke *take()* and *get_changes()* function, respectively; *loan* and *remove* message are to invoke *return_loan()* and *remove_changes()* function, respectively.

I. RTPSReader

RTPSReader process is applied to heartbeat mechanism and removing data submodule. First, it receives the heartbeat from *RTPSWriter* and sends the timely feedback to *RTPSWriter*. Then, it assists the *Subscriber* to remove the changes and data in *HistoryCache*. The behavior of *RTPSReader* is modelled as below.

```

RTPSReader() =df ComWR?msgreq.W.R.heartbeat
  → ComTC!msgreq.T.C.data.seq_num → ComCT?msgrep.C.T.complete
  → ComRW!msgrep.R.W.ACKNACK → ComDT?msgreq.D.T.remove
  → ComTD!msgrep.T.D.noinvoke2 → RTPSReader()

```

In the above formula, *RTPSReader* receives *heartbeat*, sends *data* and *seq_num* and most importantly, sends *ACKNACK* to complete the procedure of the acknowledgement mechanism.

IV. IMPLEMENTATION AND VERIFICATION

In this section, the model in Section III is implemented in the model checker PAT and the properties abstracted from the specification are all verified.

A. Implementation

First, we need to define important channels, message type flags and delivery objects as enumerations, and define messages communicated between channels as global variables. For the definition of the above variables, we give the following list as a reference:

```
channel ComPW 0;  enum {msg_req, msg_rep, msg_data};
enum {P, W, D, R, C, T, S};  var seq_num;
var ACK = 0;          var index = 0;
var DATA1;          var SEQ_NUM;
var dt[5][2];        #define HBeatInterval 5;
```

All other channels in the model are defined by the above channel format syntax like *ComPW*; the enumerated types are the type of the flag message, including *msg_req* to represent the request, *msg_rep* to stand for the reply, and *msg_data* to represent the data; *P, W, D, R, C, T, S* represent the English capital initials of the seven modules in the RPTS StatefulWriter module section. Global variable *ACK* initialized to 0 means no data received is checked by the Subscriber; global variable *index* initialized to 0 means the number of the data stored in the array in HistoryCache. *seq_num* means the initialized sequence number is zero; *DATA1* and *SEQ_NUM* are the variable representing data and sequence number in Subscriber component. Array *dt[5][2]* stores data and corresponding sequence number in HistoryCache component. Also, we give the definitions of some constant variables, for example, *HBeatInterval*, whose manual value is set to 5.

Then, we give the code of one of the processes in PAT as an example. Here we take the implementation of the *DDSWriter()* process as an example:

```
DDSWriter() = ComPW?msg_data.P.W.data1{data=data1}
→ComWR!msg_req.W.R.data → GetSeqNum();
ComRW?msg_rep.R.W.complete3{complete=complete3}
→ComWC!msg_req.W.C.data.seq_num
→ComCW?msg_rep.C.W.complete5{complete=complete5}
→ComWP!msg_rep.W.P.complete
→ComWR!msg_req.W.R.is_acked_all
→if (call(is_acked_by_all,seq_num)==1) {
ComRW?msg_rep.R.W.complete21{complete=complete21}
→ComWC!msg_req.W.C.remove
→Remove()} else {Skip};
```

From the above process execution code, it can be seen that *data1* event assigns variables and ensures variable values of all processes in the entire system are consistently changed. The function *is_acked_by_all* is invoked by *call*. *GetSeqNum()* and *Remove()* are other processes used to enhance the readability. Apparently, *GetSeqNum()* is used to get the sequence number; *Remove()* is used to remove the changes from the HistoryCache component. Their details are as follows.

```
GetSeqNum() = getSeqNum{
seq_num = seq_num + 1; } → Skip;
```

If *GetSeqNum()* is executed once, *seq_num* pluses 1, which can keep the sequence number always different and unique.

```
Remove() = atomic{
if(call(remove_change,seq_num)==1) {
ComCW?msg_rep.C.W.complete23{
complete=complete23} → Skip}
else{ ComCW?msg_rep.C.W.nocomplete23{
complete=complete-1;
complete=nocomplete23} → Skip};
```

We use *atomic* to define *Remove()* process, which means that the event cannot be disturbed until it is finished. *complete23* and *nocomplete23* are the event used to transmittting corresponding *complete* message.

Finally, the full definition of the entire system is given as follows:

```
SYSTEM() = Publisher() || DDSWriter() ||
RTPSWriter() || HistoryCache() ||
Subscriber() || DDSReader() || RTPSReader();
```

B. Properties Verification

Based on the implementation of the model in PAT above, we verify four properties as follows:

1) Divergence free

```
#assert System() divergencefree;
```

Divergence free means that any traces of the system can diverge rather than behave chaotically.

2) Consistency

Property data consistency is so important that the data from Publisher or HistoryCache or Subscriber component must be completely identical. In the implementation, the original value of the transferred data is equal to 2 and its corresponding sequence number should be equal to 1. If the data and sequence number are consistent in different components, the property is satisfied. Thus, we give the definition and assertion as follows:

```
#define goal1(dt[0][0]==1&&dt[0][1]==2)
&&(DATA1==2&&SEQ_NUM==1)
&&(dt[0][0]==SEQ_NUM&&dt[0][1]==DATA1);
#assert SYSTEM() reaches goal1;
```

3) Acknowledgement Mechanism

The reliable pattern has an acknowledgement mechanism. In our model, if the final value of the global variable *ACK* and *index* are all changed from 0 to 1, the property is satisfied. Thus, we give the LTL formula and reachability to verify whether the property is safe. Their definitions and assertions are as follows:

```
#define goal2(ACK==1&&index==1);
#assert SYSTEM() reaches goal2;
#assert SYSTEM() |=<> goal2;
```

4) Sequentiality

If the Publisher component sends several pieces of data in sequence, the pattern needs to guarantee that the data stored in the HistoryCache component must be in order. Thus, we give three atomic processes *SYSTEM02()*, *SYSTEM03()*

and *SYSTEM04()* on the basis of process *SYSTEM()*. If their storage order is correct in *HistoryCache*, the property is satisfied. Their definitions and assertions are as follows:

```
SYSTEM02() = atomic{event{data=2;} → SYSTEM();}
SYSTEM03() = atomic{event{data=4;} → SYSTEM();}
SYSTEM04() = atomic{event{data=6;} → SYSTEM();}
SYSTEM05() = SYSTEM04()||SYSTEM03()||SYSTEM02();
#define goal3 (dt[0][1] == 2&&dt[1][1] == 4&&dt[2][1] == 6);
#assert SYSTEM05() reaches goal3;
```

C. Verification and Results

According to the definitions and assertions, we implement the code in PAT and as a result, Fig. 3 shows the properties are all valid, which means the pattern of the module with no intruders is exactly reliable and also caters for the specification.

V. CONCLUSION AND FUTURE WORK

RTPS StatefulWriter module is a vital component in RTPS protocol. This paper has formalized seven components comprising the *Publisher*, *DDSWriter*, *RTPSWriter*, *DDSReader*, *HistoryCache*, *Subscriber*, *DDSReader* and *RTPSReader* with CSP. Our work also has applied the model checker PAT to implement the constructed model. Four properties abstracted from the specification, including divergence free, acknowledgement mechanism, data consistency and sequentiality, have been verified. The results are all valid. Consequently, we conclude that from the perspective of process algebra, the constructed model meets these properties and the pattern is absolutely reliable and caters for the specification.

It is naturally a great challenge to model and verify the whole RTPS protocol. We will explore security analysis and verification of the module by adding intruders in the future.

VI. ACKNOWLEDGEMENT

This work was partly supported by National Natural Science Foundation of China (Grant No. 61872145), Shanghai Collaborative Innovation Center of Trustworthy Software for Internet of Things (No.ZF1213) and Special Fund for International Academic Conferences of Graduate Students in East China Normal University.

REFERENCES

- [1] Alaerjan, A., Kim, D., Kafaf, D.A.: Modeling functional behaviors of DDS. In: 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation, SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI 2017, San Francisco, CA, USA, August 4-8, 2017. pp. 1–7 (2017)
- [2] Beckman, K., Reininger, J.: Adaptation of the DDS security standard for resource-constrained sensor networks. In: 13th IEEE International Symposium on Industrial Embedded Systems, SIES 2018, Graz, Austria, June 6-8, 2018. pp. 1–4 (2018)
- [3] Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31**(3), 560–599 (1984)
- [4] Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
- [5] Liu, Y., Guan, Y., Li, X., Wang, R., Zhang, J.: Formal analysis and verification of DDS in ROS2. In: 16th ACM/IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2018, Beijing, China, October 15-18, 2018. pp. 62–66 (2018)

Assertions	
1	SYSTEM() divergencefree
2	SYSTEM() reaches goal1
3	SYSTEM() reaches goal2
4	SYSTEM() !=> goal2
5	SYSTEM05() reaches goal3
Output	
*****Verification Result*****	
The Assertion (SYSTEM0) divergencefree) is VALID .	
*****Verification Setting*****	
Admissible Behavior: All	
Search Engine: Strongly Connected Component Based Search	
System Abstraction: False	
*****Verification Statistics*****	
Visited States:491	
Total Transitions:987	
Time Used:0.0359920s	
Estimated Memory Used:9702.912KB	
*****Verification Result*****	
The Assertion (SYSTEM0) reaches goal1) is VALID .	
The following trace leads to a state where the condition is satisfied.	
<init -> ComSD.msg_req.S.D.0 -> ComPW.msg_data.P.W.2 ->	
ComWR.msg_req.W.R.2 -> getSeqNum -> ComRW.msg_rep.R.W.1 ->	
ComRT.msg_req.R.T.0 -> ComWC.msg_req.W.C.2.1 -> ComCW.msg_rep.C.W.1	
-> ComTC.msg_req.T.C.2.1 -> ComCT.msg_rep.C.T.1 -> ComTR.msg_rep.T.R.0	
-> ComDC.msg_req.D.C.0 -> judge -> ComCD.msg_rep.C.D.1 ->	
ComDS.msg_rep.D.S.2.1->	
*****Verification Setting*****	
Admissible Behavior: All	
Search Engine: First Witness Trace using Depth First Search	
System Abstraction: False	
*****Verification Statistics*****	
Visited States:30	
Total Transitions:29	
Time Used:0.0022103s	
Estimated Memory Used:8855.672KB	
*****Verification Result*****	
The Assertion (SYSTEM0) reaches goal2) is VALID .	
The following trace leads to a state where the condition is satisfied.	
<init -> ComSD.msg_req.S.D.0 -> ComPW.msg_data.P.W.2 ->	
ComWR.msg_req.W.R.2 -> getSeqNum -> ComRW.msg_rep.R.W.1 ->	
ComRT.msg_req.R.T.0 -> ComWC.msg_req.W.C.2.1 -> ComCW.msg_rep.C.W.1	
-> ComTC.msg_req.T.C.2.1 -> ComCT.msg_rep.C.T.1 -> ComTR.msg_rep.T.R.0	
-> ComDC.msg_req.D.C.0 -> judge -> ComCD.msg_rep.C.D.1 ->	
ComDS.msg_rep.D.S.2.1 -> ComSD.msg_req.S.D.0 -> ComDT.msg_req.D.T.0 ->	
[if (((remove > 0) (remchange = 1;) else (remchange = 0;) == 0)) ->	
ComTD.msg_rep.T.D.0 -> [if ((noinvoke = 1; == 1)) -> ComDS.msg_rep.D.S.0 -> [if	
((ACKNACK == ack)) -> [if (((seq_num > 0)) (ACK = 1;) == 1)] ->	
*****Verification Setting*****	
Admissible Behavior: All	
Search Engine: First Witness Trace using Depth First Search	
System Abstraction: False	
*****Verification Statistics*****	
Visited States:55	
Total Transitions:54	
Time Used:0.0025079s	
Estimated Memory Used:8986.832KB	
*****Verification Result*****	
The Assertion (SYSTEM0) !=> goal2) is VALID .	
*****Verification Setting*****	
Admissible Behavior: All	
Search Engine: Loop Existence Checking - The negation of the LTL formula is a safety property!	
System Abstraction: False	
*****Verification Statistics*****	
Visited States:186	
Total Transitions:420	
Time Used:0.015238s	
Estimated Memory Used:10439.24KB	
*****Verification Result*****	
The Assertion (SYSTEM05) reaches goal3) is VALID .	
The following trace leads to a state where the condition is satisfied.	
<init -> eve -> ComSD.msg_req.S.D.0 -> ComPW.msg_data.P.W.2 ->	

Fig. 3. Verification Result

- [6] Lowe, G., Roscoe, A.W.: Using CSP to detect errors in the TMN protocol. IEEE Trans. Software Eng. **23**(10), 659–669 (1997)
- [7] Pérez, H., Gutiérrez, J.J.: Modeling the qos parameters of DDS for event-driven real-time applications. Journal of Systems and Software **104**, 126–140 (2015)
- [8] Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall (1997)
- [9] Roscoe, A.W.: Understanding Concurrent Systems. Texts in Computer Science, Springer (2010)
- [10] Youssef, T.A., Hariri, M.E., Elsayed, A.T., Mohammed, O.A.: A dds-based energy management framework for small microgrid operation and control. IEEE Trans. Industrial Informatics **14**(3), 958–968 (2018)

A Sound and Complete Axiomatisation for Spatio-Temporal Specification Language

Tengfei Li, Jing Liu*, Dongdong An, Haiying Sun

Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China

Abstract—Specifying spatio-temporal aspects is one of the important areas in cyber-physical systems. Spatio-temporal logic with changes of truth value in discrete time and dense time has been researched, but a combination of spatial and temporal components with changes of spatial entities in dense time hasn't been well-done. The major problem is dense time and real-valued variables of the spatio-temporal properties of cyber-physical systems. In this paper, we propose a spatio-temporal specification language, named STSL, which integrates Signal Temporal Logic (STL) with a spatial logic $S4_u$ to deal with the changes of real-values spatial entities in dense time. The combined language is divided into two formalisms, $STSL_{PC}$ and $STSL_{OC}$, which is applied to interpret the Boolean semantics and quantitative semantics, respectively. The syntax of the two formalism and the corresponding semantics are provided. Besides, we present a Hilbert-style axiomatization for the proposed STSL and provide the soundness and completeness result by the spatio-temporal extension of maximal consistent set and canonical model.

Index Terms—Signal Temporal Logic (STL) $S4_u$ Spatio-Temporal Specification Language (STSL) $STSL_{PC}$ $STSL_{OC}$ Soundness Completeness Axiomatization System

I. INTRODUCTION

It is a challenging work to model cyber-physical systems, not only because cyber-physical systems integrate cyber systems, physical environment and the interactive part of them, but also because cyber-physical systems combine temporal and spatial aspects, discrete and continuous behavior, and uncertainty. Describing spatio-temporal aspects is one of the important areas in cyber-physical systems. Many works have been done with hybrid [1] and stochastic behaviors of cyber-physical systems, but fewer researchers concentrate on spatio-temporal aspects. The major problem is multidimensional expressiveness and expensive verifiability for modeling and analysis of the spatio-temporal behaviors of cyber-physical systems.

This work aims at building a spatio-temporal specification language (STSL) by solving spatio-temporal constraints concerning dense time and real-valued variables, as an intelligent object in physical environment is provided with changes in specified space and continuous time. More specifically, we confine ourselves to the combination of topological space and time constraints with real-valued interval, which may

be an open, half-open and half-closed, half-closed and half-open, or closed interval in a flow of time. We adopt the modal spatial logic $S4_u$ to express topological constraints, which is one of the most influential formalism and the most expressiveness for topological relations. As for signal temporal logic (STL) [2], [3], there are two approaches that can cope with signals, quantitative semantics and Boolean semantics. Quantitative semantics obtains real-valued signals from *satisfaction degree* of a trace in real-valued interval. While Boolean semantics evaluates Boolean signals from a trace which can be booleanized through a set of threshold predicates.

Combining spatio-temporal constraints from temporal logics and modal spatial logics is a very important problem. Given a spatio-temporal model \mathfrak{M} and a STSL formula φ , the satisfiability problem of the formula φ is to check if φ is satisfiable in model \mathfrak{M} .

Since the changes of spatial entities and the flows of time are not independent, the combination between modal spatial logics and temporal logics is divided into two formalism, $STSL_{PC}$ and $STSL_{OC}$. $STSL_{PC}$ means the changes of spatial propositions over time, while $STSL_{OC}$ represents the changes or evolution of spatial objects over time. Each formalism is equipped with different expressiveness, so Boolean semantics and quantitative semantics need to be provided.

Many works have been done on the axiomatization and completeness of modal logics. Patrick Blackburn [4] present the completeness of normal modal logic through maximal consistent set and canonical model. J.M. Davoren [5] proposes topological semantics for intuitionistic tense logics and multi-modal logic and provide the Hilbert-style axiomatization and the completeness result. F.D. David [6] prove the absolute completeness of $S4_u$ for its measure-theoretic semantics. In this paper, we present an axiomatization system for STSL and provide the soundness and completeness result of the axiomatization system.

In this work, there are three contributions:

- 1) We propose a spatio-temporal specification language **STSL**, based on STL and $S4_u$, to specify the changes in topological space and dense time,
- 2) We interpret the **STSL** language from two formalisms: $STSL_{PC}$ and $STSL_{OC}$, and provide Boolean semantics and quantitative semantics for the language,

*Corresponding Author: jliu@sei.ecnu.edu.cn
DOI reference number: 10.18293/SEKE2019-222

- 3) We present an axiomatization system and provide the completeness result for the proposed language **STSL**.

The next section introduces temporal logic STL and modal spatial logic $S4_u$. Section 3 presents the spatio-temporal specification language **STSL** and section 4 presents an axiomatization system for the language **STSL**. In section 5, we conclude the work and talk about the future work.

II. SIGNAL TEMPORAL LOGIC AND $S4_u$

The section provides the background to the proposed spatio-temporal logic, including signal temporal logic (STL) and spatial logic $S4_u$.

A. Signal Temporal Logic (STL)

An STL signal [2], [7] is defined on dense-time domain \mathbb{T} . A *signal function* $\varepsilon: \mathbb{T} \rightarrow \mathbb{E}$ associates a set of time domain with a set of signals. Signals with $\mathbb{E} = \mathbb{B} = \{0, 1\}$ are called Boolean signals, while those where $\mathbb{E} = \mathbb{R}^+$ are called real-valued or quantitative signals. A Boolean signal, transformed from real-valued one, can be represented by Metric Temporal Logic [8], [9]. The comparison of some temporal logics is listed in Table I.

TABLE I
THE COMPARISON OF LTL, MTL AND STL

Logic	Time domain	Variable
Linear Temporal Logic (LTL)	Discrete time	Boolean value
Metric Temporal Logic (MTL)	Dense-time	Boolean value
Signal Temporal Logic (STL)	Dense-time	Real-value

An execution trace w is a set of real-valued signals x_1^w, \dots, x_k^w bound in some interval I of \mathbb{R}^+ , which is called the time domain of w [3]. Such an interval $I \subseteq \mathbb{R}^+$ is open (t_1, t_2) , half-closed and half-open $[t_1, t_2)$, half-open and half-closed $(t_1, t_2]$ or closed time interval $[t_1, t_2]$. For instance, a running car checks in and enters a highway at t_1 and checks out and leaves it at t_2 . We say, the car keeps running on the highway in the interval $[t_1, t_2]$.

The syntax of STL is given by

$$\varphi ::= \top \mid x_i \geq 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2$$

where $x_i \geq 0$ is an atomic predicate whose truth value is determined by the sign of an evaluation based on a signal x_i . The Boolean operators \neg and \wedge are negation and conjunction, respectively. The time *bounded until* operator \mathcal{U}_I is defined on the time interval I .

The formula $\varphi_1 \vee \varphi_2$, $\diamond_I \varphi$ and $\square_I \varphi$ can be defined by:

$$\begin{aligned} \varphi_1 \vee \varphi_2 &= \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\ \diamond_I \varphi &= \top \mathcal{U}_I \varphi, \\ \square_I \varphi &= \neg \diamond_I \neg\varphi. \end{aligned}$$

Formula $\diamond_I \varphi$ indicates that at some time $t \in I$, φ is eventually satisfiable, while $\square_I \varphi$ denotes that φ is always satisfiable at each time $t \in I$.

B. An axiomatisation system for STL

The fundamental composition of the axiomatization system for STL contains all the tautologies like atomic proposition, boolean and quantitative operator in first-order logic. Temporal expressiveness and inference rules are shown as follows:

A0 All classical tautologies of first-order logic

$$A1 \quad \Box(\phi \rightarrow \varphi) \rightarrow (\Box\phi \rightarrow \Box\varphi)$$

$$A2 \quad \neg \circ \phi \leftrightarrow \circ \neg \phi$$

$$A3 \quad \circ(\phi \rightarrow \varphi) \rightarrow (\circ\phi \rightarrow \circ\varphi)$$

$$A4 \quad \Box(\phi \rightarrow \circ\varphi) \rightarrow (\phi \rightarrow \Box\varphi)$$

$$A5 \quad (\phi \mathcal{U} \varphi) \leftrightarrow \varphi \vee \circ(\phi \mathcal{U} \varphi)$$

$$A6 \quad (\phi \mathcal{U} \varphi) \rightarrow \diamond\varphi$$

$$MP \quad \frac{\phi \quad \phi \rightarrow \varphi}{\varphi}$$

$$N_{\Box} \quad \frac{\phi}{\Box\phi}$$

$$N_{\circ} \quad \frac{\phi}{\circ\phi}$$

C. Spatial Logic: $S4_u$

$S4$ [10] is a proposition modal logic and τ are spatial terms under the topological space interpretation. In the absence of ambiguity, the terminology spatial terms denote spatial objects. According to an observation by [11], $S4$ is a logic of topological spaces, and the propositional variable p is interpreted as a subset of the topological space. From the perspective of the topological space, propositional variables of $S4$ will be understood as spatial variables [12]. The formula of $S4$ is defined as follows:

$$\tau ::= p \mid \bar{\tau} \mid \tau_1 \sqcap \tau_2 \mid \mathbb{I}\tau$$

where p is named as spatial variables and $\bar{\tau}$ is the complementary of τ , $\tau_1 \sqcap \tau_2$ the *intersection* operation of τ_1 and τ_2 . \mathbb{I} is an *interior* operator under the topological space interpretation. The *union* and *closure* operator can be defined by:

$$\tau_1 \sqcup \tau_2 = \overline{(\bar{\tau}_1 \sqcap \bar{\tau}_2)}, \quad \mathbb{C}\tau = \overline{\mathbb{I}\tau}$$

$\mathbb{C}\tau$ refers to the *closure* of a spatial object τ . For example, In order to ensure safety, the rear car c_{rear} can't reach the edge of the front car, the formula can be described by $\mathbb{C}_{c_{rear}} \sqcap \mathbb{C}_{c_{front}}$.

A topological model is a pair of the form $\mathfrak{M} = (\mathfrak{L}, \mathfrak{V}(p))$, where $\mathfrak{L} = (U, \mathbb{I})$ is a topological space. U is a nonempty set denoting the universe of the space, and \mathbb{I} is the interior operator on U . $\mathfrak{V}(p)$, as a set of valuations on spatial variables, is a subset of U . Therefore we get the valuation of other spatial formulas as follows:

$$\begin{aligned} \mathfrak{V}(\bar{\tau}) &= U - \mathfrak{V}(\tau), \mathfrak{V}(\tau_1 \sqcup \tau_2) = \mathfrak{V}(\tau_1) \cup \mathfrak{V}(\tau_2), \\ \mathfrak{V}(\mathbb{I}\tau) &= \mathbb{I}\mathfrak{V}(\tau), \mathfrak{V}(\tau_1 \sqcap \tau_2) = \mathfrak{V}(\tau_1) \cap \mathfrak{V}(\tau_2), \\ \mathfrak{V}(\mathbb{C}\tau) &= \mathbb{C}\mathfrak{V}(\tau). \end{aligned}$$

A spatial logic is a formal language interpreted over a class of structures featuring geometrical entities and relations. Among the well-known spatial logics such as RCC-8 [13], [14], BRCC-8 and $S4_u$, the most expressive spatial formalism is $S4_u$ [15]. $S4_u$ extends $S4$ with the *universal* and *existential* quantifiers \boxtimes , \boxdot based on a spatial term τ . $\boxtimes\tau$ refers to that

there is at least one element in space τ ; $\boxtimes \tau$ means that all elements in the space belong to τ). The formula φ is defined in the form of BNF:

$$\varphi ::= \boxtimes \tau \mid \neg \varphi \mid \varphi_1 \wedge \varphi_2$$

where, $\neg \varphi$ is the negation of φ and $\varphi_1 \wedge \varphi_2$ the conjunction of φ_1 and φ_2 . Correspondingly, the *disjunction* and *existential* operator can be defined by:

$$\varphi_1 \vee \varphi_2 = \neg(\neg \varphi_1 \wedge \neg \varphi_2), \Diamond \tau = \neg \boxtimes \bar{\tau}$$

The axiomatization system for $\mathcal{S4}_u$ includes the classical propositional logic in topology \mathcal{L} , the modal logics K, T, 4 and \neg and inference rules.

CP axioms of classical propositional logic in \mathcal{L}

$$\Box K \quad \Box(\phi \rightarrow \psi) \rightarrow (\Box \phi \rightarrow \Box \psi)$$

$$\Box T \quad \Box \phi \rightarrow \phi$$

$$\Box 4 \quad \Box \phi \rightarrow \Box \Box \phi$$

$$\neg \quad \neg \Diamond \phi \leftrightarrow \Box \neg \phi$$

and the inference rules

$$\text{MP} \quad \frac{\phi \quad \phi \rightarrow \psi}{\psi}$$

$$N_{\Box} \quad \frac{\phi}{\Box \phi}$$

$$N_{\Diamond} \quad \frac{\phi}{\Diamond \phi}$$

III. SPATIO-TEMPORAL SPECIFICATION LANGUAGE

STL provides an approach that combines the truth value and quantitative value of general signals. But it is inadequate to represent the changes of a spatial entity and the binary relation between spatial entities and temporal aspects. We propose the spatio-temporal specification language that combines STL and $\mathcal{S4}_u$ to describe the evolution in spatial and temporal domain.

A. Spatio-temporal signal

A spatio-temporal signal is defined with continuous time and topological space [16], [17]. The time domain \mathbb{T} will usually be a real-valued interval $[0, t]$, where $t \in \mathbb{R}_{\geq 0}$. The *signal function* is extended to spatial-temporal domain with $\varepsilon: \mathbb{T} \times \mathbb{L} \rightarrow \mathbb{E}$, where \mathbb{L} denotes the topological space. The domain of Boolean and quantitative signals extends the domain of STL signals to topological space.

A spatio-temporal trace $w(t, l)$ provides a notation about execution sequence of time t and space l . For a spatio-temporal trace, there are two different interpretations:

- A trace represents a sequence of spatial objects and time point and each point in the trace evaluates a pair of spatial objects and time.
- Another interpretation means that a spatio-temporal trace takes spatial objects as the basic entities and spatio-temporal primitive relations could be obtained by the changes of ontology of space over time.

In this work, we treat the spatio-temporal trace as the second interpretation. The changes of spatial objects are influenced by the flow of time.

Definition III.1 (Spatio-temporal signal). A *spatio-temporal signal* $\varepsilon(t, l)$ is an evaluation of spatial entities in a trace w over time. A *Boolean signal* $\mu(t, l)$ is an evaluation of an atomic proposition transferred from quantitative signals $\mathbf{x}(t, l)$ by atomic predicate $\mu(t, l) = (\mathbf{x}(t, l) \geq 0)$ in a trace w .

$$\varepsilon(t, l) := \begin{cases} \mu(t, l) & \text{if } \mathbb{T} \times \mathbb{L} \rightarrow \mathbb{B} \\ \mathbf{x}(t, l) & \text{if } \mathbb{T} \times \mathbb{L} \rightarrow \mathbb{R}_{\geq 0} \end{cases}$$

where boolean function $\mu: \mathbb{T} \times \mathbb{L} \rightarrow \mathbb{B}$ gives rise to the Boolean signals $\varepsilon(t, l) = \mu(t, l)$, while quantitative signals are obtained as the real-valued function $\mathbf{x}: \mathbb{T} \times \mathbb{L} \rightarrow \mathbb{R}_{\geq 0}$, with $\varepsilon(t, l) = \mathbf{x}(t, l)$.

B. The interpretation of the combined logic

It is essential that a combined spatio-temporal formalism should be provided with enough expressiveness to contain the three parameters [12]:

- 1) the expressiveness of the spatial component;
- 2) the expressiveness of the temporal component;
- 3) the interaction between the two components allowed in the combined logic.

The interaction between the spatial and temporal components should comply with the principle of *PC* and *OC*, which is used to evaluate the interaction:

- 1) $STSL_{PC}$: the language should be able to express changes over time of the truth-values of purely spatial propositions.
- 2) $STSL_{OC}$: the language should be able to express changes or evolution of spatial objects over time.

$STSL_{PC}$ expresses the change of truth-value of proposition and it is the elementary requirement for a combined spatio-temporal logic. For instance:

- * Eiffel Tower is located in Paris.
- * Two cars running on the road never occupy the same location simultaneously.

For $STSL_{OC}$, spatio-temporal properties are described about the changes of spatial objects over time. We use spatial objects in topological space to interpret the principle such as

- * One wave gradually formed in the sea and disappeared eventually on the shore.
- * If train A will pass the location in one hour that train B occupies now on the railway, train B must run within one hour.
- * A package from a courier service company will eventually be delivered to its destination through several transfer stations.

The spatio-temporal signals are divided into Boolean and quantitative signals. According to the category of spatio-temporal signals, we will present syntax and semantics for the proposed spatio-temporal specification language from two sides:

- The Boolean semantics returns true if the trace of spatio-temporal model satisfies the properties described by $STSL_{PC}$ formulas.

- The quantitative semantics returns a real value in different time that can be interpreted as an evaluation of satisfaction of the $STSL_{OC}$ formulas.

The Boolean semantics of the spatio-temporal specification language interprets that a formula of **STSL** over spatio-temporal traces returns true or false, so it is able to express changes of the truth-values concerning purely spatial propositions of PC . Meanwhile, the \mathcal{U}_I and \Box_I operators of the quantitative semantics are able to express changes or evolution of spatial objects over some fixed finite periods and the whole duration of time of OC , respectively. So, the expressiveness power of the combined spatio-temporal logic is enough to describe the spatio-temporal behavior.

C. The syntax of $STSL_{PC}$

We extend the real-value interval into spatio-temporal domain. Formally, we define the spatial temporal interval I as $[(t, l), (t', l')]$, $\forall t, t' \in \mathbb{T}$ and $t < t'$. The $STSL_{PC}$ is defined on spatio-temporal terms τ over the spatio-temporal interval I , combining temporal logic STL and modal spatial logic \mathcal{S}_{4u} . The syntax of $STSL_{PC}$ is given by:

$$\begin{aligned}\tau &::= p \mid \bar{\tau} \mid \tau_1 \sqcap \tau_2 \mid \mathbb{I}\tau \\ \varphi &::= \Box\tau \mid x_i \geq 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2\end{aligned}$$

- τ is a spatio-temporal term,
- p is a spatio-temporal variable,
- $\bar{\tau}$ is the complementary of τ ,
- $\tau_1 \sqcap \tau_2$ is the intersection of τ_1 and τ_2 ,
- \mathbb{I} is the *interior* operator under the topological space interpretation. Moreover, the dual operator of \mathbb{I} is the *closure* operator \mathbb{C} , which means possible or consistent,
- $x_i \geq 0$ is an atomic predicate,
- \neg, \vee and \wedge are the Boolean operators,
- \mathcal{U}_I is the *until* operator.

We can define equivalence of operators as syntactic abbreviations:

$$\begin{aligned}\mathbb{C}\tau &= U - \mathbb{I}\tau \\ \tau_1 \sqcup \tau_2 &= \overline{\tau_1 \sqcap \tau_2} \\ \varphi_1 \vee \varphi_2 &= \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\ \bigcirc_I \varphi &= \perp \mathcal{U}_I \varphi \\ \Diamond_I \varphi &= \top \mathcal{U}_I \varphi \\ \Box_I \varphi &= \neg \Diamond_I \neg \varphi.\end{aligned}$$

Atomic predicates, Boolean operators, and the spatio-temporal *bounded until* operator \mathcal{U}_I are from STL. The new spatial operators are the *interior* operator \mathbb{I} , the *closure* operator \mathbb{C} with reference to \mathcal{S}_{4u} . The \bigcirc_I, \Diamond_I and \Box_I operators are derived unary operators. $\Box_I \varphi$ denotes that φ holds within the whole trace of spatio-temporal interval I , and $\Diamond_I \varphi$ means that φ holds in at least one time point of the spatio-temporal interval I .

D. The semantics of $STSL_{PC}$

A spatio-temporal model is defined on topological space and temporal model. Formally, a spatio-temporal model $\mathfrak{M} = (\mathfrak{T}, \mathfrak{L}, \mathfrak{V})$, where

- \mathfrak{T} is a pair $(\mathbb{T}, <)$, where \mathbb{T} is a set of time point and $<$ \mathbb{R} an irreflexive, transitive and asymmetric relation on \mathbb{T} with a linear strict time flow,
- \mathfrak{L} is a topological space domain with the definition of U, \mathbb{I} in which U is a nonempty set, the universe of the space, and \mathbb{I} is the interior operator on U satisfying the standard Kuratowski axioms: $\forall X, Y \subseteq U, \mathbb{I}(X \cap Y) = \mathbb{I}X \cap \mathbb{I}Y, \mathbb{I}\mathbb{I}X \subseteq \mathbb{I}X$ and $\mathbb{I}(U) = U$,
- \mathfrak{V} is a valuation on the time point set \mathfrak{T} and the spatial variable set \mathbb{P} , i.e., $\forall p \in \mathbb{P}$, and $t \in \mathfrak{T}$ and $\mathfrak{U}(p, t)$ which means the space occupied by p at time point t . As for the spatial term τ , the valuation can be defined as: $\mathfrak{V}(\bar{\tau}, t) = U - \mathfrak{V}(\tau, t), \mathfrak{V}(\tau_1 \sqcap \tau_2, t) = \mathfrak{V}(\tau_1, t) \cap \mathfrak{V}(\tau_2, t), \mathfrak{V}(\mathbb{I}\tau, t) = \mathbb{I}\mathfrak{V}(\tau, t)$.

We define a spatio-temporal trace w as the changes of spatial objects over time. In spatio-temporal model \mathfrak{M} , $\mathfrak{V}(p)$ evaluates the spatial object p . Further, the spatio-temporal trace w represents $\mathfrak{V}(p)$ changes over time t , where t belongs to the domain \mathbb{T} .

The satisfaction relation for a $STSL_{PC}$ formula φ over a spatio-temporal model \mathfrak{M} is given by:

- $(\mathfrak{M}, t) \models \Box\tau \Leftrightarrow \mathfrak{V}(\Box\tau, t) = \top$
- $(\mathfrak{M}, t) \models x_i \geq 0 \Leftrightarrow \mathfrak{V}(x_i \geq 0, t) = x_i$
- $(\mathfrak{M}, t) \models \neg\varphi \Leftrightarrow (\mathfrak{M}, t) \not\models \varphi$
- $(\mathfrak{M}, t) \models \varphi_1 \wedge \varphi_2 \Leftrightarrow (\mathfrak{M}, t) \models \varphi_1$ and $(\mathfrak{M}, t) \models \varphi_2$
- $(\mathfrak{M}, t) \models \varphi_1 \mathcal{U}_I \varphi_2 \Leftrightarrow \exists t' \in t + I$ s.t. $(\mathfrak{M}, t') \models \varphi_2$ and $\forall t'' \in [t, t'], (\mathfrak{M}, t'') \models \varphi_1$

A model \mathfrak{M} satisfies φ in t , denoted by $(\mathfrak{M}, t) \models \varphi$.

For a given formula φ and execution trace w , we define the *satisfaction signal* $\chi(\varphi, w, t, l)$ over a trace $w(t, l)$:

$$\forall t \in I, \chi(\varphi, w, t, l) := \begin{cases} \top & \text{if } (\mathfrak{M}, t) \models \varphi \\ \perp & \text{otherwise} \end{cases} \quad (1)$$

In order to compute the *satisfaction* of a formula φ , we divide the formula φ into each subformula ϕ_i until atomic formula so that formula φ can be computed through the subformula and atomic formulas instead of the entire *satisfaction signal* $\chi(\varphi, w, t, l)$. The procedure can be treated as a hierarchical structure from the full formula φ down to each atomic formula.

E. $STSL_{OC}$

The difference between $STSL_{PC}$ and $STSL_{OC}$ is that $STSL_{PC}$ involves in the change of truth-values of propositions, while $STSL_{OC}$ describes the change of extensions of predicates. The syntax of $STSL_{OC}$ is given by:

$$\begin{aligned}\tau &::= p \mid \bar{\tau} \mid \tau_1 \sqcap \tau_2 \mid \mathbb{I}\tau \mid \tau_1 \mathcal{U}_I \tau_2 \\ \varphi &::= \Box\tau \mid x_i \geq 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2\end{aligned}$$

The change exists in that the operator \mathcal{U}_I can be applied to spatial terms τ . $\tau_1 \mathcal{U}_I \tau_2$ refers to τ_1 holds until τ_2 holds within the interval I . Similar to the atomic formulas, the \bigcirc_I, \Box_I and \Diamond_I of spatio-temporal term can also be derived from the operator \mathcal{U}_I . The unary operators \Box_I and \Diamond_I in $STSL_{OC}$ formulas fuse the \Box_I and \Diamond_I operators of STL with that of \Box and \Diamond operators of \mathcal{S}_{4u} .

A spatio-temporal trace w is a sequence over signals ε . The Boolean satisfaction relation and *satisfaction degree* for a $STSL_{OC}$ formula φ over a spatio-temporal trace w is similar to that of $STSL_{PC}$ formula.

We define ρ to quantify the *satisfaction degree* of the property φ over the trace $w(t, l)$, and it returns a real number $\rho(\varphi, w, t, l)$. The quantitative satisfaction relation for a formula φ over a spatio-temporal trace w is given by:

- $\rho(\Box \tau, w, t, l) = \top$
- $\rho(x_i \geq 0, w, t, l) = f(w(t, l))$ where $\mu \equiv (f \geq 0)$
- $\rho(\neg \varphi, w, t, l) = -\rho(\varphi, w, t, l)$
- $\rho(\varphi_1 \wedge \varphi_2, w, t, l) = \min\{\rho(\varphi_1, w, t, l), \rho(\varphi_2, w, t, l)\}$
- $\rho(\varphi_1 \vee \varphi_2, w, t, l) = \max\{\rho(\varphi_1, w, t, l), \rho(\varphi_2, w, t, l)\}$
- $\rho(\Box_I \varphi, w, t, l) = \inf_{t' \in t+I} \{\rho(\varphi, w, t', l')\}$
- $\rho(\Diamond_I \varphi, w, t, l) = \sup_{t' \in t+I} \{\rho(\varphi, w, t', l')\}$
- $\rho(\varphi_1 \mathcal{U} \varphi_2, w, t, l) = \sup_{t' \in t+I} (\min\{\rho(\varphi_2, w, t', l'), \inf_{t'' \in [t, t']} \{\rho(\varphi_1, w, t'', l'')\}\})$

The intuitive meaning for $\rho(\Box_I \varphi, w, t, l)$ refers to that we achieve the infimum of $\rho(\varphi, w, t', l'), \forall t' \in t + I$ over the trace. Similar to $\rho(\Box_I \varphi, w, t, l)$, $\rho(\Diamond_I \varphi, w, t, l)$ returns the supremum of $\rho(\varphi, w, t', l'), \forall t' \in t + I$.

Especially, if $x_i \geq 0$, the *satisfaction signal* will be

$$\forall t \in I, \chi(x_i \geq 0, w, t, l) := \begin{cases} \top & \text{if } x_i \geq 0 \\ \perp & \text{otherwise} \end{cases} \quad (2)$$

The connection is built between Boolean signals and quantitative signals by the way of predicate $x_i \geq 0$ and obtain the satisfaction signal $\chi(x_i \geq 0, w, t, l)$. In the quantitative semantics, however, atomic predicates $x_i \geq 0$ do not evaluate to \top or \perp but give a real value of the quantitative signals x_i by *satisfaction degree* $\rho(\varphi, w, t, l)$ representing the distance to satisfaction or not.

IV. AN AXIOMATIZATION SYSTEM FOR STSL

STSL is a logical system. A proof in **STSL** is a sequence of finite formula: A_0, A_1, \dots, A_n , where each of them is an axiom, or there exists $j, k < i$, such that A_i is the conclusion derived from A_j and A_k using *MP* inference rule. The last term A_n is a theorem in **STSL**, using the sign $\vdash A_n$, where n is the length of proof.

The notions of *deducibility* and *consistency* [4], [18] is fundamental to deduce the logic system **STSL**. A formula A is *deducible* from a set of formulas Γ in a system \mathcal{ST} , written $\Gamma \vdash_{\mathcal{ST}} A$, if and only if \mathcal{ST} contains a theorem of the form $(A_1 \wedge \dots \wedge A_n) \rightarrow A$, where the conjuncts $A_i (i = 1, \dots, n)$ of the antecedent are formulas in Γ . A set of formulas Γ is consistent in \mathcal{ST} , written $Con_{\mathcal{ST}} \Gamma$, just in case the formula \perp is not \mathcal{ST} -deducible from Γ .

Definition IV.1 (\mathcal{ST} -MCS). A set of formulas Γ is maximal \mathcal{ST} -consistent iff

- (i) Γ is \mathcal{ST} -consistent, and
- (ii) for every formula A , if $\Gamma \cup \{A\}$ is \mathcal{ST} -consistent, then $A \in \Gamma$.

If Γ is a maximal \mathcal{ST} -consistent set of formulas then we say it is an \mathcal{ST} -MCS. The (ii) condition refers to that any set of formulas properly containing Γ is \mathcal{ST} -inconsistent.

The canonical model defined by [18] to induce the soundness and completeness of modal logics. We extend the notation of *canonical model* to spatio-temporal systems for completeness of **STSL**.

Definition IV.2 (\mathcal{ST} -canonical Model). The \mathcal{ST} -canonical model \mathfrak{M}^Γ for a spatio-temporal logic is a triple $(W^\Gamma, R^\Gamma, V^\Gamma)$ where:

- (i) W^Γ is the set of all Γ -MCSs;
- (ii) R^Γ is a topological relation on topological space over a quasi-order on time. It is the canonical binary relation on W^Γ defined by $sR_i^\Gamma s'$ over state s and s' if for all formulas ϕ , $\phi \in s$ implies $\phi \in s'$.
- (iii) V^Γ is the valuation defined by $V^\Gamma(p) = \{s \in W^\Gamma \mid p \in s\}$. V^Γ is called the canonical valuation.

Lemma IV.1 (Truth Lemma). Let \mathcal{ST} -canonical model be a class of *tt-model*. For all $\phi \in \mathcal{ST}$ -MCS, $\mathcal{ST} \models \phi$ iff $\phi \in \mathcal{ST}$ -MCS.

Proof. The proof is by induction on the structure of ϕ .

Base case: Suppose ϕ is a spatial formula $\Box \tau$ or an atomic predicate $x_i \geq 0$.

$$(\mathcal{ST}, s) \models \Box \tau \Leftrightarrow V^\Gamma(\Box \tau, s) = \top \Leftrightarrow \Box \tau \in s,$$

$$(\mathcal{ST}, s) \models x_i \geq 0 \Leftrightarrow V^\Gamma(x_i \geq 0, s) = x_i \Leftrightarrow x_i \geq 0 \in s.$$

Inductive step: Suppose ϕ is an atomic predicate $\neg \phi$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\Box \phi$, $\Diamond \phi$, $\phi \mathcal{U} \varphi$. We show the proof of the case $\Box \phi$, and leave the others to reader. We have $(\mathcal{ST}, s) \models \Box \phi \Leftrightarrow \Box \phi \in s$ (assuming the inductive hypothesis).

$$(\mathcal{ST}, s) \models \Box \phi$$

$$\Leftrightarrow \forall s', sR_i^\Gamma s' \Rightarrow \mathcal{ST}, s' \models \phi$$

$$\Leftrightarrow \forall s', sR_i^\Gamma s' \Rightarrow \phi \in s'$$

we need to show that $\Box \phi \in s \Leftrightarrow \forall s', sR_i^\Gamma s' \Rightarrow \phi \in s'$.

\Rightarrow follows immediately from the definition IV.2.

As for \Leftarrow : suppose $\Box \phi \notin s$. We need to show

$$\exists s', sR_i^\Gamma s' \text{ and } \phi \notin s'$$

$$\Leftrightarrow \exists s', sR_i^\Gamma s' \text{ and } \neg \phi \in s'$$

$$\Leftrightarrow \exists s', \{\varphi \mid \Box \varphi \in s\} \subseteq s' \text{ and } \neg \phi \in s'$$

$$\Leftrightarrow \exists s', \{\varphi \mid \Box \varphi \in s\} \cup \{\neg \phi\} \subseteq s'$$

It is easy to show that $\{\varphi \mid \Box \varphi \in s\} \cup \{\neg \phi\}$ is \mathcal{ST} -consistent. Suppose not, i.e., $\{\varphi \mid \Box \varphi \in s\} \cup \{\neg \phi\}$ is \mathcal{ST} -inconsistent. Then $\vdash_{\mathcal{ST}} (\varphi_1 \wedge \dots \wedge \varphi_n) \rightarrow \phi$ for some $\{\Box \varphi_1, \dots, \Box \varphi_n\} \subseteq s$. But \mathcal{ST} is canonical and s is \mathcal{ST} -MCS, so s must contain $(\Box \varphi_1 \wedge \dots \wedge \Box \varphi_n) \rightarrow \Box \phi$. From $\Box \varphi_i \in s$, it follows $\Box \phi \in s$. This contradicts the hypothesis that $\Box \phi \notin s$. \square

Definition IV.3. The Hilbert-style proof system for the logic **STSL** has the following axiom schemes:

A0 All classical tautologies of first-order logic

A1 $\neg \circ \phi \Leftrightarrow \circ \neg \phi$

A2 $\neg \Diamond \phi \Leftrightarrow \Box \neg \phi$

- A3 $\circ(\phi \rightarrow \varphi) \rightarrow (\circ\phi \rightarrow \circ\varphi)$
A4 $\Box(\phi \rightarrow \circ\varphi) \rightarrow (\phi \rightarrow \Box\varphi)$
A5 $(\phi U \varphi) \leftrightarrow \varphi \vee \circ(\phi U \varphi)$
A6 $(\phi U \varphi) \rightarrow \Diamond\varphi$
 $\Box K$ $\Box(\phi \rightarrow \varphi) \rightarrow (\Box\phi \rightarrow \Box\varphi)$
 $\Box T$ $\Box\phi \rightarrow \phi$
 $\Box 4$ $\Box\phi \rightarrow \Box\Box\phi$

And the inference rules:

- MP $\frac{\phi \quad \phi \rightarrow \varphi}{\varphi}$
 N_{\Box} $\frac{\phi}{\vdash \Box\phi}$
 N_{\Diamond} $\frac{\phi}{\vdash \Diamond\phi}$

Soundness refers to that all the theorems in **STSL** are logically valid. Equivalently, A spatio-temporal logic is *sound* with respect to *tt*-model if for all the formula ϕ , $\vdash_{ST} \phi$ implies $\models \phi$. Let \mathcal{ST} be a class of *tt*-model, A spatio-temporal logic is *strongly complete* in \mathcal{ST} if for any set of formulas $\Gamma \cup \{\phi\}$, if $\Gamma \models_{ST} \phi$ then $\Gamma \vdash_{ST} \phi$. If the semantics of Γ satisfies ϕ on \mathcal{ST} then ϕ is deducible from Γ .

Theorem IV.2. *The above axiomatization is sound for tt-model.*

Proof. This follows from the fact that all axioms are valid and all rules preserve validity. \square

Theorem IV.3. *The system for STSL is weakly complete with respect to tt-model. i.e., for every STSL formula, $\models_{ST} \phi$ implies $\vdash_{ST} \phi$.*

Proof. We will present the proof with weak completeness of **STSL** based on the work [19], [20]. The *strong completeness* is equal to *frame completeness* and compactness in *universal modal logic* [19]. Temporal logic in the flow of real time has weak completeness [20], which proposes *finitely complete* and *expressively complete*, but fails compactness theorem. Further, a complete result based on the lexicographic products of modal logics with linear temporal logic is present in [21]. Those conclusions contribute greatly to the proof of weak completeness. \square

V. CONCLUSION AND FUTURE WORK

In this paper, we build a spatio-temporal specification language, combining STL with spatial logic $S4_u$, specifically containing dense time and topological space. We provide the syntax and semantics of the spatio-temporal language, and guarantee the seamless integration of spatial logic with temporal aspect from the perspective of the changes of purely spatial proposition $STSL_{PC}$ and spatial objects $STSL_{OC}$ over time. A Hilbert-style proof axiomatization system and the soundness and completeness result is present for the language.

The proposed **STSL** has a powerful expressiveness. It will be interesting to find a strongly complete fragment of **STSL**. For that, a more constrained axiomatization system need to be present, and more restricted inference and proof should be provided.

ACKNOWLEDGMENT

Our deepest gratitude goes to the anonymous reviewers for their valuable suggestions to improve this paper. This paper is partially supported by funding under National Key Research and Development Project 2017YFB1001800, NSFC 61572195 and Shanghai SHEITC Project 2017-GYHLW-01036.

REFERENCES

- [1] Ehsan Ahmad, Yunwei Dong, Shuling Wang, Naijun Zhan, and Liang Zou. Adding formal meanings to aadl with hybrid annex. In *International Conference on Formal Aspects of Component Software*, pages 228–247. Springer, 2014.
- [2] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: Computation and control*, pages 239–248. ACM, 2015.
- [3] Alexandre Donzé, Thomas Ferrere, and Oded Maler. Efficient robust monitoring for stl. In *International Conference on Computer Aided Verification*, pages 264–279. Springer, 2013.
- [4] Patrick Blackburn, Maarten De Rijke, and Yde Venema. *Modal Logic: Graph. Darst.*, volume 53. Cambridge University Press, 2002.
- [5] Jennifer M Davoren. Topological semantics and bisimulations for intuitionistic modal logics and their classical companion logics. In *International Symposium on Logical Foundations of Computer Science*, pages 162–179. Springer, 2007.
- [6] David Fernández-Duque. Absolute completeness of $s4_u$ for its measure-theoretic semantics. *Advances in modal logic*, 8:100–119, 2010.
- [7] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 92–106. Springer, 2010.
- [8] Patricia Bouyer. Model-checking timed temporal logics. *Electronic Notes in Theoretical Computer Science*, 231:323–341, 2009.
- [9] Haiying Sun, Jing Liu, Xiaohong Chen, and Dehui Du. Specifying cyber physical system safety properties with metric temporal spatial logic. In *Software Engineering Conference (APSEC), 2015 Asia-Pacific*, pages 254–260. IEEE, 2015.
- [10] Richard E Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM journal on computing*, 6(3):467–480, 1977.
- [11] John Charles Chenoweth McKinsey. A solution of the decision problem for the lewis systems $s2$ and $s4$, with an application to topology. *The Journal of Symbolic Logic*, 6(4):117–124, 1941.
- [12] David Gabelaia, Roman Kontchakov, Agi Kurucz, Frank Wolter, and Michael Zakharyashev. Combining spatial and temporal logics: expressiveness vs. complexity. *Journal of Artificial Intelligence Research*, 23:167–243, 2005.
- [13] David A Randell, Zhan Cui, and Anthony G Cohn. A spatial logic based on regions and connection. *KR*, 92:165–176, 1992.
- [14] Weiming Liu, Li Sanjiang, and Renz Jochen. Combining rcc-8 with qualitative direction calculi: Algorithms and complexity. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.
- [15] Roman Kontchakov, Agi Kurucz, Frank Wolter, and Michael Zakharyashev. Spatial logic+ temporal logic=? In *Handbook of spatial logics*, pages 497–564. Springer, 2007.
- [16] Laura Nenzi, Luca Bortolussi, Vincenzo Ciancia, Michele Loreti, and Mieke Massink. Qualitative and quantitative monitoring of spatio-temporal properties. In *Runtime Verification*, pages 21–37. Springer, 2015.
- [17] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.
- [18] Brian F Chellas. *Modal logic: an introduction*. Cambridge university press, 1980.
- [19] Valentin Goranko and Solomon Passy. Using the universal modality: gains and questions. *Journal of Logic and Computation*, 2(1):5–30, 1992.
- [20] Dov M. Gabbay and Ian M. Hodkinson. An axiomatization of the temporal logic with until and since over the real numbers. *Journal of Logic and Computation*, 1(2):229–259, 1990.
- [21] Philippe Balbiani and David Fernández-Duque. Axiomatizing the lexicographic products of modal logics with linear temporal logic. 2016.

Formal Specification and Model Checking of the Lim-Jeong-Park-Lee Autonomous Vehicle Intersection Control Protocol

Moe Nandi Aung

Faculty of Science

Univ. of Info. Tech. (UIT)

Hlaing Township, PO 11052, Yangon, Myanmar

Email: moenandiaung@uit.edu.mm

Yati Phyto, Kazuhiro Ogata

School of Information Science

Japan Adv. Inst. of Sci. & Tech. (JAIST)

1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

Email: {yatiphyto,ogata}@jaist.ac.jp

Abstract—We have conducted a case study in which an autonomous vehicle intersection control protocol is formally specified in Maude and model checked with Maude model checking facilities. We found that a function used in the protocol should be revised while formally specifying it and a logical clock such that times are total order should be used to avoid deadlock states during model checking experiments.

Keywords—Autonomous vehicle; Intersection control; LTL; Maude; Model checking

I. INTRODUCTION

It is predicted that vehicles would be fully autonomous (<https://www.wired.com/2012/09/ieee-autonomous-2040/>). To this end, multiple new technologies should be emerged and used. One of them is a protocol to control vehicles on an intersection such that collision must be avoided and vehicles can eventually cross the intersection. Lim, Jeong, Park & Lee have proposed such a protocol [1], which is called the LJPL protocol in this paper. Because such a protocol is intricate, formal methods should be utilized to formally verify that the protocol enjoys desired properties.

We have conducted a case study in which the LJPL protocol is formally specified in Maude [2], a rewriting logic-based specification/programming language, and model checked with the Maude model checking facilities (a reachability analyzer and an LTL model checker). While carefully reviewing the protocol, we found that one function used in the protocol should be revised. We also found that a logical clock such that times are total order should be used to avoid deadlock states during model checking experiments.

Related studies have been conducted, two among which are mentioned. Ölveczky and Meseguer [3] have reported on a case study in which a distributed embedded system (DES) family has been specified and model checked in Real-Time Maude, where the DES is for a pedestrian and car 4-way traffic intersection in which autonomous devices communicate by asynchronous message passing without a

centralized controller. Asplund, et al. [4] have demonstrated how a Satisfiability Modulo Theory (SMT) solver can be used to prove that a distributed coordination protocol for autonomous vehicles satisfies the intersection collision avoidance property. Z3 has been used as an SMT solver. Our approach uses a high abstract notion of time to formally specify the LJPL protocol, contributing to reduction of reachable state space sizes. Thus, it is unnecessary to use any realtime-related concepts and techniques.

The rest of the paper is organized as follows: § II Preliminaries, § III Lim-Jeong-Park-Lee Autonomous Vehicle Intersection Control Protocol, § IV Formal Specification, § V Model Checking, and § VI Conclusion.

II. PRELIMINARIES

A Kripke structure K is $\langle S, I, T, P, L \rangle$, where S is a set of states, $I \subseteq S$ is the set of initial states, $T \subseteq S \times S$ is a total binary relation over S , P is a set of atomic propositions and $L \in S \rightarrow 2^P$ is a labeling function. $(s, s') \in T$ is called a state transition from s to s' and T may be called the state transitions. For $s \in S$, $L(s)$ is the set of atomic propositions that hold in s . The semantics of LTL is defined over infinite sequences of states generated from K . The LTL formula used in the paper is in the form $\Diamond p$, where p is a state formula that does not have any temporal connectives. K satisfies $\Diamond p$ iff for all $s_0 \in I$, for all infinite sequencers of state starting with s_0 , there exists a state in the sequence such that p holds.

There are multiple possible ways to express states. We express a state as a braced associative-commutative (AC) collection of name-value pairs. AC collections are called soups, and name-value pairs are called observable components. That is, a state is expressed as a braced soup of observable components. The juxtaposition operator is used as the constructor of soups. Let oc_1, oc_2, oc_3 be observable components, and then $oc_1 \ oc_2 \ oc_3$ is the soup of those three observable components. A state is expressed as $\{oc_1 \ oc_2 \ oc_3\}$. There are multiple possible ways to specify state transitions. We specify them as rewrite rules. Concretely, we use Maude [2], a programming/specification

This work was partially supported by JSPS KAKENHI Grant Number JP26240008 & JP19H04082.

DOI reference number: 10.18293/SEKE2019-021

language based on rewriting logic. Maude makes it possible to specify complex systems flexibly and is also equipped with model checking facilities (a reachability analyzer and an LTL model checker). Maude allows us to use what are called matching equations in the conditional part of a conditional rewrite rule. A conditional rewrite rule (or just a rule) is in the form $\text{crl } [lb] : l \Rightarrow r \text{ if } \dots / \wedge p_1 := p_2 / \wedge \dots$, where lb is the label given to the rule and $p_1 := p_2$ is a matching equation, where p_1 may have fresh variables and p_2 does not. $p_1 := p_2$ holds if p_1 matches p_2 , making a substitution in which fresh variables in p_1 are mapped to some terms (values) in p_2 . Matching equations are similar to **let** expressions in functional programming languages.

The search command tries to find a state reachable from t such that the state matches p and satisfies c :

```
search [1] in M : t => * p such that c .
```

where M is a specification of the S and T parts of K . t typically represents an initial state of K . The search command can be used to model check that K enjoys an invariant property if p and c represent the negation of the state formula concerned.

The search command can also be used to find a deadlock state:

```
search [1] in M : t => ! p .
```

Let init be the only initial state of K and φ be an LTL formula, such as $\Diamond p$. Then, the Maude LTL model checker checks that K satisfies φ by reducing $\text{modelCheck}(\text{init}, \varphi)$.

III. LIM-JEONG-PARK-LEE AUTONOMOUS VEHICLE INTERSECTION CONTROL PROTOCOL

Let us consider the intersection shown in Fig. 1, where two streets are crossed. Vehicles are supposed to run on the right-hand side of a street and each side of a street has two lanes. Each lane is named as shown in Fig. 1. When crossing the intersection, vehicles on the right lane of one side of a street, such as lane0, are supposed to go straight or turn right as shown in Fig. 1 and those on the left lane of one side of a street, such as lane1, are supposed to turn left as shown in Fig. 1. The space overlapped by the two streets is a critical section as shown in Fig. 1, where vehicles should be controlled so that they never crash into each other.

Vehicles on lane0 and those on lane4 are allowed to go through the intersection simultaneously, while vehicles on lane0 and those on lane2 are not. lane0 and lane4 are concurrent, while lane0 and lane2 are conflict. For $i = 0, 2, 4, 6$, the conflict lanes of lane i are lane j for $j = (i + 2) \bmod 8, (i + 5) \bmod 8, (i + 6) \bmod 8, (i + 7) \bmod 8$, and the concurrent lanes of lane i are lane j for $j = (i + 1) \bmod 8, (i + 3) \bmod 8, (i + 4) \bmod 8$. For $i = 1, 3, 5, 7$, the conflict lanes of lane i are lane j for $j = (i + 1) \bmod 8, (i + 2) \bmod 8, (i + 3) \bmod 8, (i +$

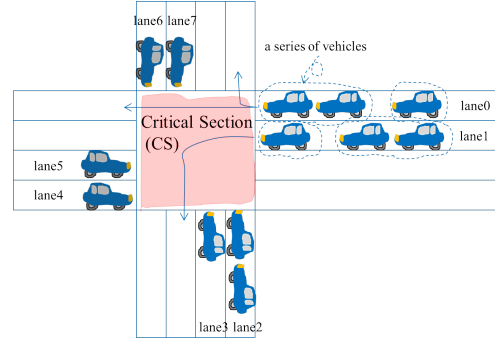


Figure 1. An intersection

$6) \bmod 8$, and the concurrent lanes of lane i are lane j for $j = (i + 4) \bmod 8, (i + 5) \bmod 8, (i + 7) \bmod 8$.

Each vehicle is given its status, which is running, approaching, stopped, crossing or crossed. Note that running and approaching are not used in the paper[1] in which the LJPL protocol is proposed. When a vehicle is far enough from the intersection¹, its status is running. Each lane is given a queue. When vehicles are approaching the intersection shortly enough, their statuses become approaching, their IDs are enqueued into a queue specific to each lane, and they never change the lane and never pass the vehicles in front of them. Vehicles then estimate the time when they are supposed to get to the intersection. The paper[1] in which the the LJPL protocol is proposed says that “To control the intersection traffic correctly, we assume that the time for vehicles is synchronized by retrieving GPS time or using a logical clock.” The present paper assumes that GPS is retrieved to synchronize vehicles, namely that there is a global clock shared by all vehicles. When a vehicle is enqueued into a queue and there is no other vehicle whose status is stopped in front on it (note that there are two such cases: (1) the vehicle is the top of the queue, namely that there is no other vehicle in front of the vehicle on the lane and (2) there is another vehicle that is followed by the vehicle and whose status is crossing on the lane), the vehicle becomes lead and its status becomes stopped. Otherwise, the vehicle becomes non-lead and its status becomes stopped. Any lead vehicle checks if there is no vehicle on any conflict lanes crossing the intersection and the time given to the vehicle is earlier than those given to the lead vehicles on the conflict lanes. If so, the lead vehicle is allowed to go through the intersection and its status becomes crossing. At the same time all non-lead vehicles whose statuses are stopped and that follow the lead vehicle are also allowed to go through the intersection and their statuses become crossing. The LJPL protocol treats a series of vehicles as a

¹When the LJPL protocol is implemented, we need to define “being far enough from,” say, 100m or longer. In this paper, however, we leave it abstract in the specification.

train so that the series of vehicles are allowed to go through the intersection simultaneously. For example, let us consider the vehicles on lane0 shown in Fig. 1 and let us suppose that the first one is lead, the second one is non-lead, the third one is lead and there is no more. When the first one is allowed to go thorough the intersection, so is the second one and then the series of the first and second ones are treated as a train.

A lead vehicle whose status is stopped on a lane exchanges some pieces of information with the lead vehicles on the four conflict lanes. One piece of information exchanged is the time (called the arrival time in this paper) when each other vehicle will arrive at the intersection. If the lead vehicle arrival time is earlier than all the four lead vehicles' arrival times on the four conflict lanes, then the status of the lead vehicle becomes crossing. Moreover, the statuses of all follower vehicles whose statuses are stopped also become crossing. When a vehicle has crossed the intersection, the status of the vehicle becomes crossed and its ID is dequeued from the queue. Such exchanges of pieces of information among lead vehicles are conducted by Algorithm 1 and Algorithm 2 [1]:

Algorithm 1. Basic IVC protocol (active thread)

```

1: begin at each round
2:    $vehicle_{target} \leftarrow selectVehicle();$ 
3:    $send(information_{local}, vehicle_{target});$ 
4:    $information_{target} \leftarrow receive(vehicle_{target});$ 
5:    $updateInformation(information_{local},$ 
                      $information_{target});$ 
6: end

```

Algorithm 2. Basic IVC protocol (passive thread)

```

1: repeat
2:    $vehicle_{target} \leftarrow waitForVehicle();$ 
3:    $information_{target} \leftarrow receive(vehicle_{target});$ 
4:    $updateInformation(information_{local},$ 
                      $information_{target});$ 
5:    $send(information_{local}, vehicle_{target});$ 
6: until forever;

```

where IVS stands for inter-vehicle-communication [5]. $information_x$, where $x = target, local$, consists of the following pieces of information:

- $lane$ – Lane number from 0 to 7
- $arrivalTime$ – Arrival time for its own vehicle
- $arrivalTime_{lead}$ – Arrival time for lead vehicle
- $lead$ – True or false
- $conflictLane$ – List of conflict lanes
- $concurrentLane$ – List of concurrent lanes
- $concurrentLanePassing$ – List of concurrent lanes for passing vehicles
- $status$ – stopped, passing, or passed

Note that running and approaching are also used as status values in this paper.

The LJPL protocol itself is described as Algorithm 3 [1]:

Algorithm 3. Mutual exclusion algorithm via IVC

```

1: begin initialization
2:    $infoVehicles_i[j] \leftarrow null, \forall i \in \{1, \dots, max_{lane}\},$ 
    $\forall j \in \{1, \dots, max_{vehicle}\};$ 
3: end
4: begin when entering the intersection
5:    $lane \leftarrow getLaneNum();$ 
6:    $arrivalTime \leftarrow getCurrentTime();$ 
7:   if no vehicle on the lane,
   where  $status == stopped$  then
8:      $lead \leftarrow true;$ 
9:      $arrivalTime_{lead} \leftarrow arrivalTime;$ 
10:  else
11:     $lead \leftarrow false;$ 
12:  endif
13:   $status \leftarrow stopped;$ 
14: end
15: begin at each cycle
16:   update  $infoVehicles_i[j]$ 
   according to Algorithm 1 and Algorithm 2;
17:   check  $infoVehicles_{conflictLane}$ 
   for passing the intersection;
18:   if passingCondition() then
19:      $status \leftarrow passing;$ 
20:     move and cross the intersection;
21:   endif
22: end
23: begin when exiting the intersection
24:    $status \leftarrow passed;$ 
25: end
26: function passingCondition()
27:   if  $\forall arrivalTime_{lead} \in infoVehicles_{conflictLane}$ 
    $> arrivalTime_{lead}$  and
    $\forall status \in infoVehicles_{conflictLane}$ 
    $== stopped$  then
28:     return true;
29:   else if  $\exists status \in infoVehicles_{concurrentLane}$ 
    $== passing$  and
    $\exists ! lane_i \in \{lane_n, \forall n \in \{0, \dots, max_{lane}\}$ 
    $| status == passing\}$  and
    $\forall arrivalTime_{lead}$ 
    $\in infoVehicles_{concurrentLanePassing}$ 
    $> arrivalTime_{lead}$  then
30:     return true;
31:   else
32:     return false;
33:   end function

```

From a vehicle v point of view such that v is located on a lane l , function passingCondition() returns true even if there exists a lead vehicle v' on a conflict lane of l such that the status is passing (namely that the first condition is

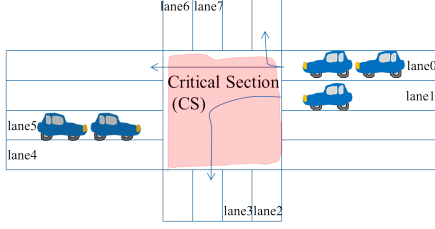


Figure 2. An initial state

false) but if the second condition is true. If so, v is permitted to cross the intersection and may crash into v' . Therefore, `passingCondition()` is revised as follows: it returns true if the first condition is true and returns false otherwise.

IV. FORMAL SPECIFICATION

Let K_{LJPL} be the Kripke structure formalizing the LJPL protocol.

The four kinds of observable components are used:

- $(v[vid] : lid, vstat, t, lt)$ – vid is a vehicle ID (a natural number), lid is a lane ID (a natural number) on which the vehicle vid is located, $vstat$ is the status of the vehicle vid , t is the time when the vehicle vid will reach the intersection and lt is the time when the lead vehicle will reach the intersection if any.
- $(lane[lid] : q)$ – lid is a lane ID (a natural number) and q is a queue of vehicle IDs (natural numbers).
- $(clock : t, b)$ – t is a natural number that represents an abstract notion of the current time and b is a Boolean value. If time is allowed to increase without any constraints, the reachable state space can quickly explode. Therefore, the following constraint will be put: whenever b is true, t can increment and b becomes false if so. When a vehicle obtains the current time t (which does not change t , though), b becomes true.
- $(gstat : gstat)$ – $gstat$ is either `fin` or `nFin`. If it is `fin`, then all vehicles concerned have crossed the intersection.

Let us consider the initial state as shown in Fig. 2 such that two vehicles are running on lane0, one vehicle is running on lane1, two vehicles are running on lane5 and there is no vehicle on the other lanes. The initial state (referred as `init`) is expressed as follows:

```
{(gstat: nFin) (clock: 0, false)
 (lane[0]: oo) (lane[1]: oo) (lane[2]: oo)
 (lane[3]: oo) (lane[4]: oo) (lane[5]: oo)
 (lane[6]: oo) (lane[7]: oo)
 (v[oo]: 0, stopped, oo, oo) (v[oo]: 1, stopped, oo, oo)
 (v[oo]: 2, stopped, oo, oo) (v[oo]: 3, stopped, oo, oo)
 (v[oo]: 4, stopped, oo, oo) (v[oo]: 5, stopped, oo, oo)
 (v[oo]: 6, stopped, oo, oo) (v[oo]: 7, stopped, oo, oo)
 (v[0]: 0, running, oo, oo) (v[1]: 0, running, oo, oo)
 (v[2]: 1, running, oo, oo) (v[3]: 5, running, oo, oo)
 (v[4]: 5, running, oo, oo) }
```

Each queue for each lane, such as the one represented by $(lane[0] : oo)$, only consists of `oo` that represents ∞ , which means that there is no vehicle on the lane close enough to the intersection. There are eight $v[oo]$ observable components that are used to represent dummy vehicles. The $v[0]$, $v[1]$, $v[2]$, $v[3]$, and $v[4]$ observable components represent the five vehicles, the first two of which are on lane0, the third of which is on lane1 and the last two of which are on lane5. The global clock represented by $(clock : 0, false)$ is initially 0. Because the second value of the `clock` observable component is `false`, the abstract notion of the current time cannot increment. The value of the `gstat` observable component is initially `nFin`.

T_{LJPL} is specified by 12 rewrite rules. The following rule is used to make T_{LJPL} total:

```
rl [stutter] : {(gstat: fin) OCs}
=> {(gstat: fin) OCs} .
```

where `OCs` is a Maude variable of observable component soups.

```
cr1 [fin] : {(gstat: nFin) OCs}
=> {(gstat: fin) OCs} if fin?(OCs) .
```

where `fin?(OCs)` returns true if and only if each vehicle in `OCs` has crossed the intersection.

```
rl [tick] :
{(gstat: nFin) (clock: T, true) OCs}
=>
{(gstat: nFin) (clock: (T + 1), false) OCs} .
```

where T is a Maude variable of natural numbers. If the second value of the `clock` observable component is `true`, the abstract notion of the current time T increments and the second value becomes `false`.

Two rules are used to specify a set of transitions that change a vehicle status from running to approaching. One rule deals with the case in which there is no vehicle close enough to the intersection on the lane where the vehicle is running, and the other deals with the case in which there exists at least one vehicle close enough to the intersection on the lane. The two rules are as follows:

```
rl [approach1] :
{(gstat: nFin) (clock: T, B) (lane[LI]: oo)
 (v[VI]: LI, running, oo, oo) OCs}
=>
{(gstat: nFin) (clock: T, true)
 (lane[LI]: VI) (v[VI]: LI, approaching, T, oo)
 OCs} .

rl [approach2] : {(gstat: nFin) (clock: T, B)
 (lane[LI]: (VI' ; VS))
 (v[VI]: LI, running, oo, oo) OCs}
=> {(gstat: nFin) (clock: T, true)
 (lane[LI]: (VI' ; VS ; VI))
 (v[VI]: LI, approaching, T, oo) OCs} .
```

where B is Maude variable of Boolean values, LI , VI & VI' are Maude variables of natural numbers, VS is a

Maude variable of queues of natural numbers & ∞ , and $_;$ is the constructor of queues, where an underscore $_$ is a place holder where an argument is put. $_;$ is associative, meaning that $(q_1 ; q_2) ; q_3 = q_1 ; (q_2 ; q_3)$, and a single element (a natural number or ∞) is treated as the singleton queue that only consists of the element.

Three rules are used to specify a set of transitions that change a vehicle status from approaching to stopped. The first rule `check1` deals with the case in which the vehicle is the top of the queue concerned, where the vehicle will be lead on the lane concerned, the second rule `check2` deals with the case in which there exists a vehicle in front of the vehicle on the lane concerned such that the preceding vehicle status is stopped, where the vehicle will be non-lead on the lane, and the third rule `check3` deals with the case in which there exists a vehicle in front of the vehicle on the lane concerned such that the preceding vehicle status is crossing, where the vehicle will be lead on the lane. The three rules are as follows:

```

r1 [check1] :
{(gstat: nFin) (lane[LI]: (VI ; VS))
 (v[VI]: LI,approaching,T,oo) OCs}
=> {(gstat: nFin) (lane[LI]: (VI ; VS))
 (v[VI]: LI,stopped,T,T) OCs} .

r1 [check2] : {(gstat: nFin)
 (lane[LI]: (VS' ; VI' ; VI ; VS))
 (v[VI']: LI,stopped,T,T')
 (v[VI]: LI,approaching,T'',oo) OCs}
=> {(gstat: nFin)
 (lane[LI]: (VS' ; VI' ; VI ; VS))
 (v[VI']: LI,stopped,T,T')
 (v[VI]: LI,stopped,T'',T') OCs} .

r1 [check3] : {(gstat: nFin)
 (lane[LI]: (VS' ; VI' ; VI ; VS))
 (v[VI']: LI,crossing,T,T')
 (v[VI]: LI,approaching,T'',oo) OCs}
=> {(gstat: nFin)
 (lane[LI]: (VS' ; VI' ; VI ; VS))
 (v[VI']: LI,crossing,T,T')
 (v[VI]: LI,stopped,T'',T') OCs} .

```

where T' & T'' are Maude variables of natural numbers and VS' is a Maude variable of queues. The reason why $v[vid]$ observable components, where vid is a vehicle ID, do not have any values that correspond to *lead* in *information_x* is that it is possible to use queues, etc. to manage which vehicles are lead or not.

Two rules are used to specify a set of transitions that change a lead vehicle status from stopped to crossing. One rule deals with the case in which the ID of the lane on which the lead vehicle is located is even and the other deals with the case in which it is odd. The first rule `enter1` is as follows:

```

cr1 [enter1] :
{(gstat: nFin) (lane[LI]: (VI ; VS))
 (v[VI]: LI,stopped,T,T) OCs}

```

```

=> {(gstat: nFin) (lane[LI]: (VI ; VS))
 (v[VI]: LI,crossing,T,T) OCs'}
if isEven(LI) /\
  LI1 := (LI + 2) rem 8 /\
  (lane[LI1]: (VI1 ; VS1))
  (v[VI1]: LI1,VSt1,T11,T12) OCs1 := OCs /\
  VSt1 = stopped /\ T < T12 /\
  LI2 := (LI + 5) rem 8 /\
  (lane[LI2]: (VI2 ; VS2))
  (v[VI2]: LI2,VSt2,T21,T22) OCs2 := OCs /\
  VSt2 = stopped /\ T < T22 /\
  LI3 := (LI + 6) rem 8 /\
  (lane[LI3]: (VI3 ; VS3))
  (v[VI3]: LI3,VSt3,T31,T32) OCs3 := OCs /\
  VSt3 = stopped /\ T < T32 /\
  LI4 := (LI + 7) rem 8 /\
  (lane[LI4]: (VI4 ; VS4))
  (v[VI4]: LI4,VSt4,T41,T42) OCs4 := OCs /\
  VSt4 = stopped /\ T < T42 /\
  OCs' := letCross(VS,OCs) .

```

where LI_i for $i = 1, \dots, 4$ is a Maude variable of natural numbers, VI_i & T_j for $i = 1, \dots, 4$ & $j = 11, 12, \dots, 41, 42$ are Maude variables of natural numbers & ∞ , VS_i for $i = 1, \dots, 4$ is a Maude variable of queues, VSt_i for $i = 1, \dots, 4$ is a Maude variable of vehicle statuses, and OCs_i & OCs' for $i = 1, \dots, 4$ are Maude variables of observable component soups. `isEven(LI)` holds if LI is even. For the front-most vehicle (which may be a dummy one whose ID is `oo`) of each of the four conflict lanes of lane LI , the rules checks if it is not crossing the intersection and its arrival time is greater than the arrival time T of the vehicle VI concerned. If the conditions are fulfilled, the status of the vehicle VI concerned becomes crossing from stopped and the statuses of all vehicles that follow VI and whose statuses are stopped also become crossing, which is done by `letCross(VS,OCs)`. The second rule `enter2` can be described likewise.

Two rules are used to specify a set of transitions that change a vehicle status to crossed from crossing. The first rule deals with the case in which the vehicle concerned is one and only one vehicle in the queue that corresponds to the lane concerned, and the second rule deals with the case in which the queue that corresponds to the lane concerned contain two or more vehicles. The two rules are as follows:

```

r1 [leave1] :
{(gstat: nFin) (lane[LI]: VI)
 (v[VI]: LI,crossing,T,T') OCs}
=> {(gstat: nFin) (lane[LI]: oo)
 (v[VI]: LI,crossed,T,T') OCs} .

r1 [leave2] :
{(gstat: nFin) (lane[LI]: (VI ; VI' ; VS))
 (v[VI]: LI,crossing,T,T') OCs}
=> {(gstat: nFin) (lane[LI]: (VI' ; VS))
 (v[VI]: LI,crossed,T,T') OCs} .

```

When a vehicle status changes to closed from crossing, it is deleted from the queue that corresponds to the lane concerned.

V. MODEL CHECKING

The paper [1] in which the LJPL protocol is proposed takes into account three desired properties the protocol should enjoy:

- *Safety (version 2)* No vehicles in conflict lanes are in CS at the same time.
- *Deadlock-freedom* If a vehicle is trying to pass the intersection (CS), then some vehicle, not necessarily the same one, finally pass the intersection.
- *Starvation-freedom* If a vehicle is trying to pass the intersection (CS), then the vehicle must finally pass the intersection in finite time.

The *Safety (version 2)* property can be checked by the following search command:

```
search [1] in IMUTEX : init =>*
{ (v[VI]: LI, crossing, T, T')
  (v[VI']: LI', crossing, T2, T2') OCS }
such that areConflict(LI, LI') .
```

where IMUTEX is the specification of the protocol and `areConflict(LI, LI')` holds if and only if the lanes LI and LI' are conflict. The search commands tries to find a state from the initial state `init` such that two vehicles are crossing the intersection on two conflict lanes LI and LI'. It does not find any such states, meaning that the protocol enjoys the *Safety (version 2)* property for the case as shown in Fig. 2.

The *Deadlock-freedom* property can be checked by the following search command:

```
search [1] in IMUTEX : init =>! {OCS} .
```

The search commands tries to find a state from the initial state `init` such that no transition can be conducted. It finds such a state that contains the following observable components:

```
(lane[0]: 0 ; 1) (lane[5]: 3 ; 4)
(v[0]: 0, stopped, 0, 0) (v[1]: 0, stopped, 0, 0)
(v[3]: 5, stopped, 0, 0) (v[4]: 5, stopped, 0, 0)
```

Vehicle 0 is the lead one on lane0, while vehicle 3 is the lead one on lane5. The both lead vehicles' arrival times are 0. Therefore, either one is not permitted to cross the intersection.

The counterexample of the *Deadlock-freedom* property implies that it does not suffice to use a global clock, which may create a symmetry. To break the symmetry, lane IDs could be used. When there are multiple lead vehicles on conflict lanes such that their arrival times are exactly the same, higher priorities are given to those on lanes whose IDs are less. That is, a logical clock such that times are total order is used. In the specification, the two rules `enter1` and `enter2` should be revised such that $T < T1i$ for $i = 1, \dots, 4$ used in the conditions is replaced with $(T < T1i \text{ or } (T == T1i \text{ and } LI < LIi))$. The protocol in which a logical clock such that times are

total order is used enjoys both the *Safety (version 2)* and *Deadlock-freedom* properties for the case as shown in Fig. 2. Let the protocol refer to the one in which a logical clock such that times are total order is used.

To model check that the protocol enjoys the *Starvation-freedom* property, it is necessary to define P_{LJPL} and L_{LJPL} . P_{LJPL} consists of one atomic proposition `fin`. L_{LJPL} is defined as follows:

```
eq {(gstat: fin) OCS} |= fin = true .
eq {OCS} |= PROP = false [otherwise] .
```

where PROP is a Maude variable of atomic propositions. The two equations say that for all states $s \in S_{LJPL}$ $L_{LJPL}(s) = \{\text{fin}\}$ if and only if s contains `(gstat: fin)`. The *Starvation-freedom* property is then defined as follows:

```
eq halt = <> fin .
```

where `<>` is the LTL eventually connective \Diamond . We model check if the protocol enjoys the *Starvation-freedom* property by reducing `modelCheck(init, halt)`. No counterexample is found. Thus, the protocol enjoys the property for the case as shown in Fig. 2.

VI. CONCLUSION

We have reported on a case study in which the LJPL protocol is formally specified in Maude and model checked with Maude model checking facilities. We found that a function used in the protocol should be revised while formally specifying it and a logical clock such that times are total order should be used to avoid deadlock states during model checking experiments.

We have encountered the state explosion problem when we tried to model check that the protocol enjoys some property for the case as shown in Fig. 1. One piece of our future work is to remedy the situation, say, by using the divide & conquer approach to (leads-to) model checking [6].

REFERENCES

- [1] J. Lim, Y. Jeong, D. Park, and H. Lee, "An efficient distributed mutual exclusion algorithm for intersection traffic control," *The Journal of Supercomputing*, vol. 74, pp. 1090–1107, 2018.
- [2] Clavel, M., et al., *All About Maude*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4350.
- [3] P. C. Ölveczky and J. Meseguer, "Specification and verification of distributed embedded systems: A traffic intersection product family," in *RTRTS 2010*, 2010, pp. 137–157.
- [4] M. Asplund, A. Manzoor, M. Bouroche, S. Clarke, and V. Cahill, "A formal approach to autonomous vehicle coordination," in *FM 2012*, 2012, pp. 52–67.
- [5] M. L. Sichitui and M. Kihl, "Inter-vehicle communication systems: A survey," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 1-4, pp. 88–105, 2008.
- [6] Y. Phyo and K. Ogata, "Formal specification and model checking of the Walter-Welch-Vaidya mutual exclusion protocol for ad hoc mobile networks," in *APSEC 2018*, 2018.

Improving the Applicability of the Ranked Nodes Method to build Expert-Driven Bayesian Networks

João Nunes^{*1}, Luiz Silva^{†1}, Mirko Perkusich^{‡1}, Kyller Gorgonio^{§1}, Hyggo Almeida^{¶1}, and Angelo Perkusich^{||1}

¹Embedded and Pervasive Computing Laboratory, Federal University of Campina Grande, Campina Grande, Brazil

Abstract

One challenge in constructing a Bayesian network (BN) is defining the node probability tables (NPTs), which can be learned from data or elicited from domain experts. In practice, for large-scale BN it is common not to have enough data for learning and elicitation from experts is unfeasible. Previous work proposed a solution to this problem: the Ranked Nodes Method (RNM). However, this solution needs to be applied by a RNM expert who, through the elicitation of expert judgement, identifies the necessary parameters for the RNM algorithm to generate the NPTs. Hence, this paper presents a novel approach to define NPT using the RNM with no ranked nodes-specific knowledge. The solution is named Simulated Bayesian Network Expert (SBNE). It consists of eliciting a subset of the NPT from the domain experts which is used as input to an algorithm that estimates the optimal parameters for the RNM to generate the NPTs. To validate our solution, we conducted an experiment with multiple domain experts and compared the results with other methods. Our solution outperformed the other methods (producing NPTs at least 12% more accurate) and is, therefore, a promising approach to apply RNM without relying on RNM experts.

Bayesian networks; expert systems; knowledge acquisition; ranked nodes.

^{*}joao.bezerra@embedded.ufcg.edu.br

[†]luiz.silva@embedded.ufcg.edu.br

[‡]mirko.perkusich@embedded.ufcg.edu.br

[§]kyller@embedded.ufcg.edu.br

[¶]hyggo@embedded.ufcg.edu.br

^{||}perkusic@embedded.ufcg.edu.br

1 Introduction

Bayesian Network (BN) is a mathematical model that graphically and numerically represents the probabilistic relationships between random variables through Bayes theorem. Recently, given the evolution of the computational capacity, which enabled the calculation of complex BNs, it has become a popular technique to assist on decision-making [7] and it has been applied in several areas such as large-scale engineering projects [12], software engineering [16, 14], and sports management [4].

The challenges for the construction of BNs can be divided into two sub-problems: (i) construct the directed acyclic graph (DAG) and (ii) define the NPTs. In this research, we focus on (ii). In cases where there is historical data with enough information about the domain to be modelled it is possible to automate the process of NPT definition through batch learning [10].

Unfortunately, in practice, in most cases, there is not enough data [7] to apply batch learning. In such cases, it is necessary to manually define the NPTs through the elicitation of domain experts knowledge. However, given that the complexity of building NPTs grows exponentially, depending on the number of parents and states, the manual definition of the NPT becomes unfeasible.

To reduce the complexity of manually defining a NPT through the elicitation of knowledge from domain experts, Fenton et al. [7] proposed the Ranked Nodes Method (RNM). This method is limited to nodes (i.e., random variables) with an ordinal scale (e.g., “Good”, “Medium”, “Bad”), which are called ranked nodes.

In ranked nodes, the ordinal scale is mapped into a scale monotonically ordered in the interval $[0, 1]$. The solution is based on a Normal distribution truncated between $[0, 1]$

(i.e., TNormal) to represent the NPTs. Hence, the NPT of a child node is a TNormal calculated as the mixture of the TNormals of its parent nodes. There are four expressions to model the mixture's mean (μ): weighted mean (*WMEAN*), weighted minimum (*WMIN*), weighted maximum (*WMAX*) and the mixture of the classic minimum and maximum functions (*MIXMINMAX*).

Hence, to properly use the RNM it is necessary to understand the mixture process to select the appropriate parameters: the weighted expression; a set of weights of the parent nodes; and the variance (σ). However, even when RNM experts are available, the application of the RNM method still presents challenges.

The means to identify a suitable expression to mix the TNormals of the parent nodes based on mode assessments of the domain experts is straightforward, as described in Laitila and Virtanen [11]. Conversely, the discovery of the weights and variance parameters are far more complex. Such tasks are usually performed by the RNM experts using a trial and error strategy. This strategy comes down to a cycle of generating, verifying and adjusting the parameters to regenerate the NPTs, which is repeated until a satisfying result is discovered [7, 11].

In this paper, we present a novel approach to improve the applicability of RNM. Our main goal is to encapsulate its complexity, allowing for its use by domain experts with no prior knowledge about ranked nodes. We use “what if” analysis (i.e., truth tables) to elicit knowledge from experts using visual aids. Given the information collected, we use the expert system proposed in Silva et al. [5] to obtain the weighted expression to mixture the TNormals and an algorithm named “Simulated Expert” to estimate the optimal variance and set of weights of the parent nodes.

We evaluated our solution with an experiment in which multiple domain experts applied our approach to RNM and two other methods to quantify a BN model related to the evaluation of cohesion of agile software development teams. The NPTs generated with the three methods were compared in terms of accuracy using manually defined NPTs as benchmark. The results showed that our solution is promising as it has achieved greater accuracy compared to the other methods.

2 Background

BNs are probabilistic graph models used to represent knowledge about uncertain domains. A BN, B , is a directed acyclic graph that represents a joint probability distribution over a set of random variables V [9]. The network is defined by the pair $B = \{G, \Theta\}$. G is the directed acyclic graph in which the nodes X_1, \dots, X_n represent random variables and the arcs represent the direct dependencies between these variables. Θ represents the set

Table 1. Example of a truth table.

Parent A	Parent B	Parent C	Child D
Very low	Very high	Very low	Low
Very high	Very low	Very low	Low
Very low	Very low	Very high	Low
Very low	Very high	Very high	High
Very high	Very low	Very high	Low
Very high	Very high	Very low	Medium

of probability functions. This set contains the parameter $\theta_{x_i|\pi_i} = P_B(x_i|\pi_i)$ for each x_i in X_i conditioned by π_i , the set of parents of X_i in G . Equation 1 presents the joint distribution defined by B over V .

$$P_B(X_1, \dots, X_n) = \prod_{i=1}^n P_B(x_i|\pi_i) = \prod_{i=1}^n \theta_{X_i|\pi_i} \quad (1)$$

We present an example of a BN in Fig. 1, in which ellipses represent the nodes and arrows represent the arcs. The probability functions are usually represented by NPTs. Even though the arcs represent the causal connection's direction between the variables, information can propagate in any direction [13].

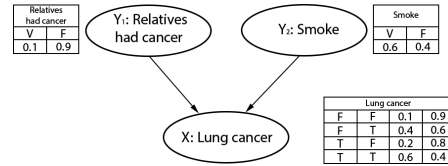


Figure 1. A Bayesian network example.

According to Fenton et al. [7], to define the NPT, the RNM user should define the resulting TNormal parameters constructing “truth tables” using example scenarios, which they define as “what if” analysis. An example is shown in Table 1. By analysing it, we can conclude that defining the parameters is not straightforward and there is a need to understand the TNormal mixture process to apply the RNM.

In Perkusich et al. [15], a simplified approach to use the RNM whenever there is a need to collect data from multiple experts was presented. Instead of using “what if” analysis, it asks the experts to order the relationships between the child and parents nodes given their relative magnitude. The collected data is analysed statistically and used to define the weights for the function of μ , having the function type defined to *WMEAN* and a fixed variance of $5.0E^{-4}$. Therefore, although it encapsulates the complexity of the RNM approach, it has limited modelling capabilities and, as discussed in Perkusich et al. [16], it might produce incorrect NPTs.

In da Silva et al. [5], an approach based on production rules was proposed to encapsulate the complexity of calibrating the NPTs. Given a set of input values, the developed expert system automatically calibrates the NPTs. In this work, the modelling capabilities of the approach presented in Fenton et al. [7] is combined with ranked nodes specific knowledge encapsulation of the approach presented in Perkusich et al. [16]. However, the proposed approach also fixed the variance in $5.0E^{-4}$, which is a limiting factor.

3 Solution

SBNE is an approach to elicit expert knowledge and apply the RNM method without relying on the assistance of RNM experts. SBNE stands for Simulated Bayesian Network Expert. This approach can be divided into three steps: (i) direct or indirect probability assessment from domain experts; (ii) use of production rules¹ to define the weighted expression; and (iii) use of the “Simulated Expert” algorithm to estimate the input parameters required to apply the RNM method. In (i) domain experts use a GUI (still a prototype) that allows them to evaluate probability distributions directly (i.e., using numbers), or indirectly (i.e., using a visual tool).

3.1 Knowledge Elicitation Process

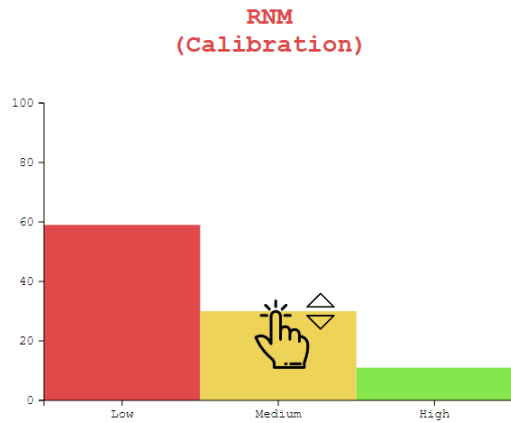


Figure 2. Component Prototype (A).

The prototype used in the probability elicitation process is here decomposed into two separate figures (for presentation purposes only). Hence, for each combination of extreme cases of the parent nodes (i.e., each row in the truth table) domain experts provides the expected probability distribution using an interactive bar chart (see Fig. 2) or sliders horizontally arranged (see Fig. 3). During

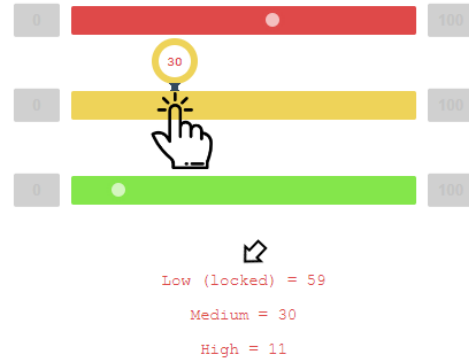


Figure 3. Component Prototype (B).

the elicitation process the domain experts directly interacts with the vertical bars so that by raising or lowering one of the bars the others automatically adjust itself. Strictly speaking, the natural order, which would be to inform the probability distributions with numbers and update the bar chart, is subverted so that users can interact and evaluate probability distributions by reasoning in terms of proportion rather than numerical terms, if they so wish. The sliders shown in Fig. 3 behave the same way.

A commonly employed strategy by domain experts is to first set up the bar relative to the state that they have greater confidence in estimating, and adjust the other states accordingly. To apply this strategy, the users can lock or unlock states by clicking over it, as shown with an arrow in Fig. 2, so that changes in other states do not modify the locked states. In short, the domain experts are able to provide the input data to the “Simulated Expert”, reasoning in terms of proportion, in which case they ignore the numerical information of the prototype, or reasoning directly in numerical terms, in which they use the elements presented in Fig. 3.

That been established, let us consider a simple case in which we have a child node C with two parent nodes, A and B, all having three states each (e.g., “low”, “medium” and “high”). To generate the child node’s NPT, the domain expert needs to assess four probability distributions as shown in Table 2. The Table 2 is basically a truth table composed by all the combinations of extreme states of the parent nodes.

In this case, each row in Table 2 is filled with data from the interaction of the domain experts with the components presented in Fig. 2 and 3. In other words, the domain experts inform four probability distributions, which constitutes a subset of the child node’s NPT. This subset is then used as input for the “Simulated Expert” to estimate the optimal parameters for the RNM algorithm. The weighted expression is defined using production rules as proposed in [5].

¹Files available at <https://github.com/SEKE2019/SBNE>

Table 2. Truth table for a node with two parents.

Rows	Parents		Child		
	A	B	C		
			Low	Medium	High
1	Low	Low	1	0	0
2	Low	High	0	0.3	0.7
3	High	Low	0	0.3	0.7
4	High	High	0	0	1

3.2 Simulated Expert

The Simulated Expert is an algorithm that receives as input the target probability distributions and a weighted expression to estimate the most suitable parameters (i.e., variance and set of weights of the parents) for the RNM algorithm to generate the NPTs.

The algorithm can be divided into three steps: (i) search for the most likely range of the optimal variance; (ii) identify the combination of weights of the parent nodes; and (iii) estimate the optimal variance parameter, as detailed below.

Step (i):

1. generate a variance vector V in range $5.0E^{-4}$ to 0.2 with step $5.0E^{-4}$;
2. set the weight of all parent nodes to 5;
3. define a “resolution” constant $\delta = 10$ in which the desired accuracy is inversely proportional to its value;
4. calculate the step $s = \frac{|V|}{\delta}$ to perform the search for the most probable interval of the optimal variance;
5. traverse V and at each s , define the variance of the child node using the current variance (e.g., shaded boxes in Fig. 4), calculate the NPT and its relative score. Do that until there is no room for improvement (e.g., row 5 in Fig. 4) or until it reaches the end of V ;
6. when the score starts decreasing return to the previous used variance index and calculate $a = index - \frac{|V|}{\delta}$ and $b = index + \frac{|V|}{\delta}$, the most likely interval where the optimal variance must be (e.g., row 7 in Fig. 4).

Step (ii):

1. set the variance of the child node using the median variance in $v \in V$ (i.e., $v = V[a : b]$), the interval at which the optimal variance is more likely to be (e.g.,

Table 3. Truth table approximation from the Simulated Expert.

Rows	Parents		Child		
	A	B	C		
			Low	Medium	High
1	Low	Low	0.9965	0.0035	0
2	Low	High	0	0.2995	0.7005
3	High	Low	0	0.2995	0.7005
4	High	High	0	0.0035	0.9965

Result obtained with the Simulated Expert for the child node C using the weighted expression $WMAX$ with the following parameters: $\sigma^2 = 0.062$; weight 4 for all parent nodes; $BS = 3.94E^{-06}$.

output of (i) illustrated in Fig. 4 row 7 in which it would be the variance located in index 5);

2. runs all combinations of weights of the parent nodes and stores the optimal set of weights.

Step (iii):

1. using the optimal set of weights obtained in previous step, traverse the subset v (e.g., row 7 in Fig. 4) performing the same actions as in item 5 from step (i).

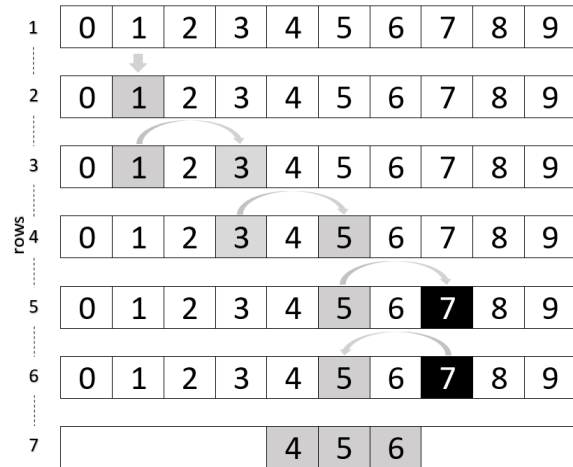


Figure 4. Illustration of the execution of the algorithm from step (i).

For each execution of the RNM algorithm, a score of the estimated probabilities relative to the target probabilities is computed using the Brier Score, but any other similarity measure can be used (e.g., euclidean distance, Kullback-Leibler divergence).

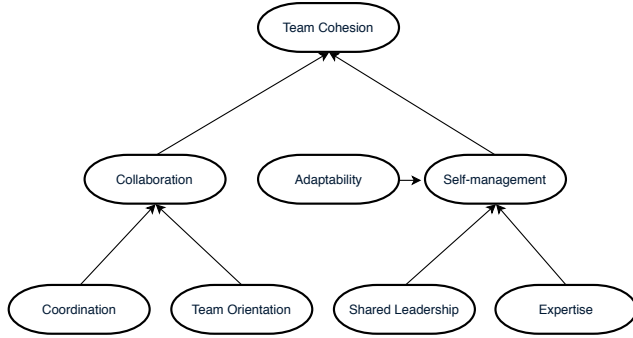


Figure 5. BN used in the experiment.

4 Empirical Validation

An experiment was conducted with undergraduate students that work as junior software developers at the Embedded Lab – a software development lab located at the Federal University of Campina Grande, Brazil. The purpose of the experiment was to validate the proposed approach.

In the experiment, 10 developers quantified a BN (adapted from the model proposed in Freire et al. [8]) using the proposed approach and two others methods: the Weighted Sum Algorithm (WSA) and a variation of the WSA, here named WSAAHP, that employs indirect probability elicitation by means of pairwise comparisons between states in which the domain experts compare the likelihood of states using a verbal and numerical scale.

Each domain expert also manually defined NPTs that served as benchmark for comparing the methods. A Randomised Complete Block Design (RCBD) was adopted in the experiment. The methods were compared in terms of accuracy. The accuracy is defined here as how well the NPTs generated represent the mode of probability distributions in the benchmark NPTs.

We focused on the following research question and null hypothesis:

RQ1: Does the use of the SBNE approach to the RNM method maintain, improve or degrades expert-driven BNs accuracy?

H₀: The proposed approach to RNM is less accurate than the other methods.

The BN used in the experiment is presented in Fig. 5. The domain experts built 40 NPTs. Nevertheless, only the three parent nodes NPT was considered in the analysis (i.e., the one associated with the child node self-management), since it is the most complex. All the NPTs are available in an online repository².

²<https://github.com/SEKE2019/SBNE>

Table 4. Tukey simultaneous tests for differences of means

Difference of Method Levels	Difference of Means	SE of Difference	Simultaneous 95% CI	T-Value	Adjusted P-Value
WSA-RNM	-0.1267	0.0414	(-0.2323; -0.0211)	-3.06	0.018
WSAAHP-RNM	-0.1602	0.0414	(-0.2658; -0.0546)	-3.87	0.003
WSAAHP-WSA	-0.0335	0.0414	(-0.1391; 0.0721)	-0.81	0.702

Individual confidence level = 98%.

An analyse of variance (ANOVA) was performed, which indicated that there is statistically significant difference (p-value = 0.003) between the accuracy of the methods with a significance level of 0.05. Tukeys HSD post hoc test was performed to determine which methods are in fact different in regards to accuracy level.

The Table 4 summaries the Tukey simultaneous test for differences of means. As can be seen in the Table 4, the confidence interval for the difference between the means of WSA-RNM and WSAAHP-RNM do not include zero, which indicates that the difference is statistically significant between these methods. The results show that RNM is 13% and 16% (i.e., rounded values) more accurate than WSA and WSAAHP, respectively. Therefore, H₀ was rejected.

5 Threats to Validity

Despite the results obtained with the proposed approach, the comparison with different methods poses as a threat to internal validity. Nevertheless, the method WSA can be considered as a good benchmark because it has been mathematically and empirically validated in the literature [6, 2]. Moreover, the external validity may be limited, considering that the participants of the experiment were undergraduate students who work as software developers and that the experiment is bound to a specific context.

6 Conclusions

Despite recent popularity, the construction of BNs is still challenging. One of the challenges refers to defining the NPTs for large-scale BN. It is possible to automate this process using batch learning when there is a database with enough information. In practice, this is not common. The other option is to elicit data from experts, which becomes unfeasible for large scale BN. Fenton et al. [7] presented a solution based on ranked nodes. However, to apply this solution a BN expert is usually necessary.

In this paper, we complement the work of Fenton et al. [7] and Silva et al. [5] by presenting a novel approach to apply the RNM without relying on BN experts. The solution is named Simulated Bayesian Network Expert (SBNE) and it consists of eliciting a subset of the NPT from the domain

experts, which is used as input to estimate the optimal parameters (without relying on RNM experts) for the RNM to generate the NPTs.

Nonetheless, this approach can be used by RNM experts, reducing their effort to identify the optimal parameter for the RNM algorithm. We compared the proposed approach to the RNM with two other methods and the RNM outperformed them, reaching a mean accuracy of 75.78% against 63.12% of WSA and 59.77% of WSAHP. These results are promising and validate our approach, which makes RNM accessible to a wider range of users.

Notwithstanding, it is not our goal to state which method is the best. For such a purpose, more experiments would be needed to investigate the matter. That said, it is our belief that future works in this area should concentrate on examining the proposed approach against BNs derived from RNM experts and comparing multiple methods using well known BN models from the literature such as ALARM [3] and Hailfinder [1].

References

- [1] B. Abramson, J. Brown, W. Edwards, A. Murphy, and R. L. Winkler. Hailfinder: A bayesian system for forecasting severe weather. *International Journal of Forecasting*, 12(1):57–71, 1996.
- [2] S. Baker and E. Mendes. Evaluating the weighted sum algorithm for estimating conditional probabilities in bayesian networks. In *SEKE*, volume 2010, pages 319–324, 2010.
- [3] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. In *AIME 89*, pages 247–256. Springer, 1989.
- [4] A. C. Constantinou, N. E. Fenton, and M. Neil. Profiting from an inefficient association football gambling market: Prediction, risk and uncertainty using bayesian networks. *Knowledge-Based Systems*, 50:60 – 86, 2013.
- [5] R. M. da Silva, M. Perkusich, R. M. Saraiva, A. S. Freire, H. O. Almeida, and A. Perkusich. Improving the applicability of bayesian networks through production rules. In *SEKE*, pages 8–13, 2016.
- [6] B. Das. Generating conditional probabilities for bayesian networks: Easing the knowledge acquisition problem. *arXiv preprint cs/0411034*, 2004.
- [7] N. E. Fenton, M. Neil, and J. G. Caballero. Using ranked nodes to model qualitative judgments in bayesian networks. *IEEE Trans. on Knowl. and Data Eng.*, 19(10):1420–1432, Oct. 2007.
- [8] A. Freire, M. Perkusich, R. Saraiva, H. Almeida, and A. Perkusich. A bayesian networks-based approach to assess and improve the teamwork quality of agile teams. *Information and Software Technology*, 100:119–132, 2018.
- [9] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.
- [10] D. Heckerman. Learning in graphical models. chapter A Tutorial on Learning with Bayesian Networks, pages 301–354. MIT Press, Cambridge, MA, USA, 1999.
- [11] P. Laitila and K. Virtanen. Improving construction of conditional probability tables for ranked nodes in bayesian networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(7):1691–1705, 2016.
- [12] E. Lee, Y. Park, and J. G. Shin. Large engineering project risk management using a bayesian belief network. *Expert Syst. Appl.*, 36(3):5880–5887, Apr. 2009.
- [13] J. Pearl and S. Russell. Bayesian networks. *Handbook of brain theory and neural networks*, 1995.
- [14] M. Perkusich, K. Gorgonio, H. Almeida, and A. Perkusich. Assisting the continuous improvement of scrum projects using metrics and bayesian networks. *Journal of Software: Evolution and Process*, 2016. Article in Press.
- [15] M. Perkusich, A. Perkusich, and H. O. de Almeida. Using survey and weighted functions to generate node probability tables for bayesian networks. In *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 183–188. IEEE, 2013.
- [16] M. Perkusich, G. Soares, H. Almeida, and A. Perkusich. A procedure to detect problems of processes in software development projects using bayesian networks. *Expert Systems with Applications*, 42(1):437 – 450, 2015.

A systematic process to define expert-driven software metrics thresholds

Renata Saraiva¹, Mirko Perkusich¹, Hyggo Almeida¹, and Angelo Perkusich¹

¹Embedded and Pervasive Computing Laboratory, Federal University of Campina Grande, Campina Grande, Brazil

Abstract

Software metrics are usually used for quantification, not giving the necessary support for decision making. To increase their usefulness, it is necessary to give them meaning through the definition of significant thresholds. Despite its importance, the state of the art on threshold derivation is mostly based on data-driven approaches. This paper presents a systematic approach to define thresholds for metrics in the absence of data and based on eliciting knowledge from experts. The proposed approach is based on identifying context factors that influence the thresholds for a given metric and is supported by fuzzy logic concepts to model the crisp value (i.e., collected data) into a linguistic variable (i.e., interpreted information). We present context factors elicited from three experts for the metrics code coverage, static code analysis warnings count and defect count. Further, we present cases on how to implement the proposed approach. As a result, we conclude that the approach is promising.

Keywords—Software metrics; Software thresholds; Fuzzy logic.

I. Introduction

Despite their potential advantages, software metrics are usually used only for quantification purposes, not giving adequate support for decision-making [1]. For this purpose, it is necessary to give meaning (i.e., semantics) to the metrics through the definition of reference values (i.e., thresholds). Thresholds are values used to set ranges of desirable and undesirable states and indicate anomalies. The absence of thresholds for many software metrics is one of the main reasons for them to not be effectively used in the industry [4].

There are several solutions to calculate software metrics thresholds proposed in the literature [1], [18], [17], [8], [4], [23], [15], [13], [14], [5]. We summarize their characteristics and limitations in Section II. As far as our knowledge, all the proposed solutions are data-driven and focus on source code metrics. For instance, recently, two empirical studies were published comparing the existing data-driven thresholds models to predict faults on open source software [2] and fault proneness [3]. Therefore, if an organization needs to define thresholds for popular management metrics such as *team velocity*, *build status* and *lead time* [9], the literature is scarce in guiding them on how to define representative thresholds for their context. For this purpose, in practice, organizations use expert knowledge to define thresholds using ad hoc processes. For instance, for the metric *code coverage*, many development groups require 85% coverage to achieve quality targets as the status quo [20]. On the other hand, there are several factors that might influence coverage target for a given project, such as product complexity, project criticality, and cost of evolution.

To complement the current state of the art on software metrics threshold derivation, we present a systematic approach to define thresholds for metrics in the absence of data and based on eliciting knowledge from experts. The goal is to support managers when making decisions regarding the interpretation (i.e., semantics) of the collected metrics by developing models that mimics the thought process of humans when making decisions regarding the thresholds.

The proposed approach is based on identifying context factors that influence the thresholds for a given metric and is supported by fuzzy logic concepts to model the crisp value (i.e., collected data) into a linguistic variable (i.e., interpreted information). For instance, consider that the metric *code coverage* is used to decide if enough tests have been executed and, given that the *defect count* is

low enough, the product can be delivered to the customer. The crisp value of *code coverage* lies in $[0, 100]$. So, for instance, we could map the value 85 to the linguistic variable *OK*, which means that, given this metric, the product should be released. Conversely, we could map the value 50, meaning that the product should not be released.

The research question that we address in this paper is:

How can we define thresholds for software metrics in the absence of data and when the organization context cannot be reduced to an experimental setup?

Our contributions include: (1) a systematic approach for deriving software metrics thresholds in the absence of data; (2) an empirical cyclic process to continuously refine thresholds; and (3) context factors that influence the definition of popular metrics such as *code coverage*, *static code analysis warnings count* and *defect count*. Our systematic approach is demonstrated through the definition of thresholds for popular software metrics.

The remaining of the paper is organized as follows: in Section II, we present works related to deriving thresholds and discuss them in light of our approach, the proposed process; in Section III, we present the proposed approach; and in Section IV, we present our final remarks and future work.

II. Related work

The effective use of software metrics is hampered by the lack of significant thresholds [1]. In the literature, few metrics have defined thresholds. Furthermore, many researchers have proposed different approaches to define them [1], [4], [5], [13], [15], [17], [18], [23].

Alves *et al.* [1] present a method that determines threshold empirically from measurement data (i.e., benchmarking). The method is based on statistical properties of the metric such as scale and distribution. To evaluate their approach, they collected data from 100 object-oriented software systems to calculate thresholds, which were successfully used to assist on software analysis, benchmarking and certification. The main risk of such a solution is to use thresholds to assist decision-making that were calculated for a different context.

In the works of Oliveira *et al.* [15], [13], the concept of relative thresholds is proposed as well as a tool for extracting these thresholds. Their approach handles the heavy-tailed distribution of source code metrics by complementing absolute thresholds with a percentage of software code entities that must follow it. The technique is validated with an industrial case study. As Alves *et al.* [1], its limitation is that the calculated threshold and percentage might be dependent on the context.

Ferreira *et al.* [4] used the EasyFit tool to define the probability distribution with the best fit for the distribution

for a given metric. Therefore, if the defined probability distribution had a representative mean value, it was used as the reference value. Otherwise, the distribution is quantified as bad, good or moderate. By analyzing data of forty open source projects, they defined the thresholds for six metrics: LCOM (Lack of Cohesion of Methods), DIT (Depth in Tree), COF (Coupling Factor), afferent couplings, number of public methods, and number of public fields.

In Foucault *et al.* [5], a solution based on statistical methods was presented. This approach is based on (i) *double sampling* [19] to randomly selects projects samples, and (ii) *bootstrap* to estimate the thresholds based on quartiles. Despite the potential of this approach, the validation process was limited to a test to identify the best configuration for the approach itself since, according to the authors, the two statistical methods are widely used.

In Shatnawi [17], a solution based on logarithmic transformation was presented. In this approach, initially, the data is transformed using the natural log, leaving the symmetric data thus closer to a normal distribution. Afterward, a temporary reference value (T') is collected using the mean (M) and standard deviation (SD) so that $T' = M + SD$ or $T' = M - SD$. Finally, the T' is converted to the original distribution by using the exponent function of T' , generating the final reference value.

All the presented studies are data-driven and most of them focus on source code metrics. The motivation of our work is to define a systematic process that guides engineers in defining metrics thresholds in the absence of data. In this context, Marinescu [10] presents a guideline to define semantical filtering to support the analysis of source code metrics in the context of detecting design flaws based on the derivation of thresholds. They define two types of thresholds-related filters: marginal and interval. For a marginal filter, it is necessary to define the threshold value and direction, which specifies whether the threshold value is an upper or lower bound. The thresholds are described as design rules or heuristics (e.g., a class should not be coupled with more than 6 other classes). Interval filters are defined as an interval such as “between 20 and 30”. Even though Marinescu [10] describes the use of thresholds as a key component on the proposed solution, he does not present a systematic approach to define it. As shown in Section III, we use the classification of thresholds presented in Marinescu [10] and complement their work by proposing a systematic approach to define them.

III. Threshold definition strategy

In this section, we present the proposed approach with running examples. The examples are a result of a pilot study executed at a Brazilian software development

company in which we collected data from 3 project managers, all of them with over 5 years of experience managing projects with support of software metrics. The most important decision they had was defining if a version of the product had enough quality to be released. The three main metrics they used for this purpose were *code coverage*, *static code analysis warnings count* and *defect count*. *Code coverage* was used to indicate if enough tests were performed, which gave them confidence that few defects would be detected only in operation. *Static code analysis warnings count* was used as a measure of the internal quality of the product. The *defect count*, which was only representative if *code coverage* was high enough, gave a snapshot of the current quality of the product. As a status quo on the company, all projects had a lower bound threshold of 80% for *code coverage*, 10 for *static code analysis warnings count*, and 5 for *defect count*.

The main goal of the proposed approach is to provide software engineers with a systematic mechanism to enable them to work with software metrics in a more abstract level through the definition of thresholds in the absence of historical data. Since the goal of using metrics is to support decision-making, this is closer to the real intention in using metrics. An assumption of the proposed approach is that the metrics are valid for their intended purposes [12].

The proposed expert-driven threshold definition strategy is cyclic and composed of three main steps, namely: (i) thresholds characterization, (ii) thresholds modeling, and (iii) thresholds evaluation. An overview of the approach is shown in Figure 1. Our approach can be used to elicit data from a single or multiple experts. There are two main roles: threshold designer and domain experts. The threshold designer is responsible for leading the planning and execution of the threshold definition process by the elicitation of knowledge from the domain experts. The domain experts are responsible for actively participating in the process of defining the thresholds models (steps i and ii) and evaluating the models (steps iii). The domain experts might include the project manager, development lead, test lead, product manager or quality assurance manager.

In what follows, we present details regarding each of the steps of the proposed approach.

A. Step i: thresholds characterization

On the first (i) step, the goal is to characterize the thresholds through the identification of the relevant context factors and the type of the threshold, as defined in Marinescu [10]. First, it is necessary to define the decision scale to be used.

Definition 1 (Metric semantics scale) A metrics semantic scale is the scale to be used to represent the

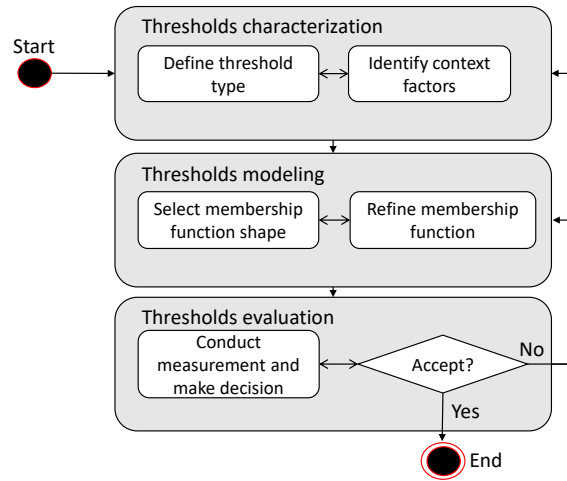


Fig. 1. Proposed approach overview.

semantics of a given metric.

The possible types of scale are Boolean and ordinal. For instance, one may use a Boolean scale for *code coverage* with the values: *OK* and *NOT OK*. Another possibility is to use an ordinal scale: *Bad*, *Moderate* and *Good*.

Afterwards, the type of threshold must be defined: marginal or interval. If a marginal threshold is selected, there are two possible types: **HigherThan**(Θ) and **LowerThan**(Θ), where Θ is the reference value. If an interval threshold is selected, by definition is of the type *Between*(α, β), which is equivalent to **HigherThan**(α) \wedge **LowerThan**(β), where α is the lower bound and β is the upper bound.

Next, the context factors must be identified. Context factors are attributes from key entities of the process that might influence the threshold. There are two types of context factors: diminishing factors and enhancing factors. The diminishing factors influence the thresholds values to be lower, and the enhancing factors influence the thresholds values to be higher. For instance, for *code coverage* metric, we elicited the following factors from the experts: *product complexity* and *project criticality* as enhancing factors, while *team experience* and *impact on evolution cost* as diminishing factors. For *static analysis warnings count* and *defect count*, *project criticality* is the enhancing factor. Finally, the factors are ranked in order of relative magnitude of their influence on the thresholds' values.

It is important to notice that the context factors are factors that influence the semantics of the metrics and not the value itself. For instance, one might reason that the factor "volatile requirements" influences the threshold of "defect

count”, because the more volatile are the requirements, less time will be available for testing and, consequently, lower will be the “defect count” in the testing phase (not necessarily, the “defect leakage”). On the other hand, even though this might be true, the reference value for the given metric should not be lower, which discards this factor.

In the case that multiple domain experts are involved in the process, the threshold designer must execute the described tasks with each one individually to avoid bias. Afterwards, with all the domain experts together, he presents the identified possibilities of threshold types and context factors and executes a meeting like the Planning Poker [7], in which each expert holds two cards: Agree and Disagree. The threshold designer mediates the meeting by enabling a structured discussion regarding a consensus of the threshold types and context factors to be considered in the models. As in a Planning Poker meeting, candidate solutions must be individually voted by experts, by turning their card simultaneously, until a consensus is reached by all the experts.

B. Step ii: thresholds modeling

After defining the type of threshold and rank the context factors, the experts will have a better understanding regarding the semantics of the given metrics. Since our approach is based on concepts of fuzzy logic, the thresholds are modeled as a linguistic variable, which is “a variable whose values are words or sentences in a natural or artificial language” [22]. For this purpose, we map the metric semantic scales defined in step i as the term set to be used for the linguistic variable. So, for instance, we could have the linguistic variable *code coverage* (c) composed of the terms {OK, Not OK}, in the case of a Boolean scale.

The main goal of this step is to fuzzify the crisp values of a metric into fuzzy linguistic terms. For this purpose, it is necessary to define the membership functions. There are several types of membership functions that can be used. In Figure 2, we show six popular types of functions.

A membership function must be defined for each term in the given linguistic variable. So, for a metric with a Boolean scale, two membership functions must be defined.

The main challenge is to define the parameters for the membership functions. For this purpose, given the type of function and the magnitude of the impact of the context factors identified in step i, the threshold designer might show the experts possible shapes for the membership functions to guide them. For instance, if the threshold is marginal, probably the experts could choose a z-shape, sigmoid, or s-shape as a reference. If the threshold is interval, the experts could choose triangular, trapezoidal or Gaussian. For instance, for the *code coverage*, we could use the Gaussian shape.

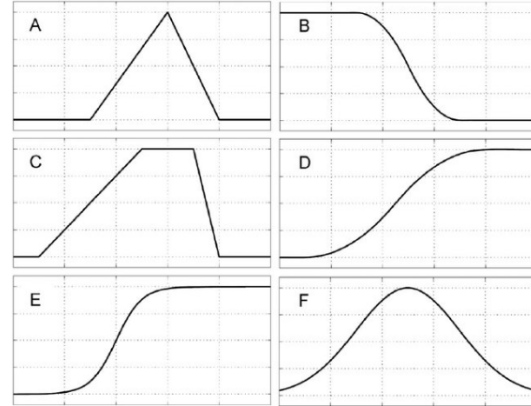


Fig. 2. Six types of fuzzy membership functions: (A) triangular, (B) z-shape, (C) trapezoidal, (D) s-shape, (E) sigmoid and (F) Gaussian, [6].

TABLE I. “What if” scenarios for *Code coverage* with a verbal scale

Code coverage crisp value	Not OK	OK
10	Certain	Impossible
20	Certain	Impossible
30	Certain	Impossible
40	Probable	Improbable
50	Expected	Uncertain
60	Fifty-fifty	Fifty-fifty
70	Uncertain	Expected
80	Improbable	Probable
90	Uncertain	Expected
100	Probable	Improbable

Afterwards, the experts should use their experience from past projects to define “what-if” scenarios to guide them in configuring the functions, in which for a set of values, they would indicate the probability of it being mapped to each of the possible terms. For this purpose, instead of directly defining the probabilities, they could use a verbal scale such as the one presented in Renooij and Witteman [16]. For instance, for *code coverage*, the scenarios shown in Table I could be used:

Afterwards, the verbal scale is converted to a numerical scale following the rules presented in Renooij and Witteman [16]. As a result, we have the values presented in TableII.

Given this, an algorithm can be used to fit the data elicited from the experts for each linguistic term into the appropriate distribution. For instance, in Figure 3, we present a fit for the membership function for the term *Not OK* using the Akima Cubic Spline. Finally, the expert can analyze visually the resulting distribution and judge if it reflects his intuition (i.e., face validity). Otherwise, they must reflect on the inconsistencies and the step should restart.

TABLE II. “What if” scenarios for *Code coverage* with a numerical scale

Code coverage crisp value	Not OK	OK
10	100	0
20	100	0
30	100	0
40	85	15
50	75	25
60	50	50
70	25	75
80	15	85
90	25	75
100	85	15

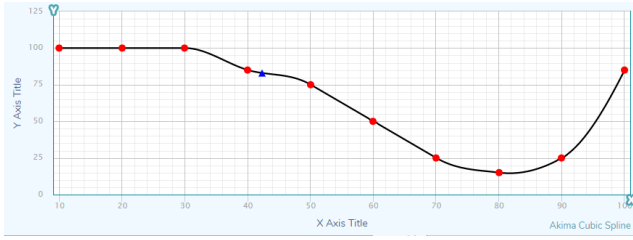


Fig. 3. Curve fit for *Code Coverage*'s linguistic term *Not OK*.

As a rule of thumb, it is preferable to set thresholds that are more conservative, since it is better to get more false positive results, rather than missing an important issue due to a very strict threshold value. The threshold can be refined during the empirical cycle which is executed on step iii.

As in step ii, this step should be, initially, executed individually with each expert. Afterwards, consensus must be achieved between all the experts in a Planning Poker-style meeting.

C. Step iii: thresholds evaluation

At this point, the thresholds models are defined, but as in other expert-driven processes [11], [21] it is necessary to have an empirical cycle in which decisions based on the thresholds are analyzed to evaluate the models. For this purpose, the threshold designer should schedule meetings according to the project's context. For agile projects with short-term releases, a meeting could be held every iteration or two. During the meeting, along with the threshold designers, the domain experts that participated on steps i and ii should participate to discuss the results of using the models.

Assuming that the metrics are valid for their intended purposes, if the model's results are not consistent with the reality, there are three possible outcomes: (1) exception case, (2) scope limitation, and (3) model needs refinement. For the first outcome, it is possible that, for instance,

the developed model indicates with 90% that the product should be released, but the release is a failure. This might occur due to a rare case for the given organization, which might be the case of an unexpected success of the product causing overload on the server. In this case, the experts could decide that the model is reliable, and the bad decision was caused by the uncertainty inherent in the process.

For the second possible outcome, a bad decision of releasing a product might have been caused by a failure of requirements elicitation such as missing an important non-functional requirement (e.g., number of requests per second). For this case, the experts can assume that the decisions based on the model assume that the product's requirements are complete and that this case is out of the scope of the model.

For the third case, the experts might decide that the model needs refinement, because they failed to, for instance, consider an important context factor or a better suited membership function. Independent of the outcome, the execution of step iii should be considered a mandatory activity in the measurement program.

IV. Conclusions

In this paper, we presented a systematic process to define thresholds for metrics in the absence of data and based on eliciting knowledge from experts. The proposed approach is based on identifying context factors that influence the thresholds for a given metric and is supported by fuzzy logic concepts to model the crisp value (i.e., collected data) into a linguistic variable (i.e., interpreted information). It is cyclic and composed of three main steps, namely: (i) thresholds characterization, (ii) thresholds modeling, and (iii) thresholds evaluation.

Our contributions are threefold: (1) a systematic approach for deriving software metrics thresholds in the absence of data; (2) an empirical cyclic process to continuously refine thresholds; and (3) context factors that influence the definition of popular metrics such as *code coverage*, *static code analysis warnings count* and *defect count*.

For further research, we will expand the proposed approach to handle the case of having multiple metrics used for a single decision. Furthermore, we will execute a case study to empirically evaluate the proposed approach in terms of practical utility.

References

- [1] T. L. Alves, C. Ypma, and J. Visser. Deriving metric thresholds from benchmark data. In *2010 IEEE International Conference on Software Maintenance*, pages 1–10. IEEE, sep 2010.

- [2] O. F. Arar and K. Ayan. Deriving thresholds of software metrics to predict faults on open source software. *Expert Syst. Appl.*, 61(C):106–121, Nov. 2016.
- [3] A. Boucher and M. Badri. Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. *Information and Software Technology*, 96:38–67, 2018.
- [4] K. A. Ferreira, M. A. Bigonha, R. S. Bigonha, L. F. Mendes, and H. C. Almeida. Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, 85(2):244–257, feb 2012.
- [5] M. Foucault, M. Palyart, J.-R. Falleri, and X. Blanc. Computing contextual metric thresholds. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*, pages 1120–1125, New York, New York, USA, mar 2014. ACM Press.
- [6] M. U. Guide. Matlab cd-rom, mathworks, 2007.
- [7] N. C. Haugen. An empirical study of using planning poker for user story estimation. In *AGILE 2006 (AGILE'06)*, pages 9–pp. IEEE, 2006.
- [8] S. Herbold, J. Grabowski, and S. Waack. Calculation and optimization of thresholds for sets of software metrics. *Empirical Software Engineering*, 16(6):812–841, 2011.
- [9] E. Kupiainen, M. V. Mäntylä, and J. Itkonen. Using metrics in agile and lean software development—a systematic literature review of industrial studies. *Information and Software Technology*, 62:143–163, 2015.
- [10] R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, pages 350–359. IEEE, 2004.
- [11] E. Mendes. Expert-based knowledge engineering of bayesian networks. In *Practitioner's Knowledge Representation*, pages 73–105. Springer, 2014.
- [12] A. Meneely, B. Smith, and L. Williams. Validating software metrics: A spectrum of philosophies. *ACM Trans. Softw. Eng. Methodol.*, 21(4):24:1–24:28, Feb. 2013.
- [13] P. Oliveira, F. P. Lima, M. T. Valente, and A. Serebrenik. Rttool: A tool for extracting relative thresholds for source code metrics. In *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 629–632. IEEE, 2014.
- [14] P. Oliveira, M. T. Valente, A. Bergel, and A. Serebrenik. Validating metric thresholds with developers: An early result. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 546–550. IEEE, 2015.
- [15] P. Oliveira, M. T. Valente, and F. P. Lima. Extracting relative thresholds for source code metrics. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 254–263. IEEE, feb 2014.
- [16] S. Renooij and C. Wittman. Talking probabilities: communicating probabilistic information with words and numbers. *International Journal of Approximate Reasoning*, 22(3):169 – 194, 1999.
- [17] R. Shatnawi. Deriving metrics thresholds using log transformation. *Journal of Software: Evolution and Process*, 27(2):95–113, feb 2015.
- [18] R. Shatnawi, W. Li, J. Swain, and T. Newman. Finding software metrics threshold values using ROC curves. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(1):1–16, jan 2010.
- [19] S. K. Thompson. Simple random sampling. *Sampling, Third Edition*, pages 9–37, 2012.
- [20] T. Williams, M. Mercer, J. Mucha, and R. Kapur. Code coverage, what does it mean in terms of quality? In *Annual Reliability and Maintainability Symposium. 2001 Proceedings. International Symposium on Product Quality and Integrity (Cat. No. 01CH37179)*, pages 420–424. IEEE, 2001.
- [21] B. Yet, Z. Perkins, N. Fenton, N. Tai, and W. Marsh. Not just data: A method for improving prediction with knowledge. *Journal of Biomedical Informatics*, 48:28 – 37, 2014.
- [22] L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning—i. *Information sciences*, 8(3):199–249, 1975.
- [23] F. Zhang, A. Mockus, Y. Zou, F. Khomh, and A. E. Hassan. How Does Context Affect the Distribution of Software Maintainability Metrics? In *2013 IEEE International Conference on Software Maintenance*, pages 350–359. IEEE, sep 2013.

Multi-source fault detection and diagnosis based on multi-level Knowledge Graph and Bayesian theory reasoning

Tao Sun

School of Computer Science and Technology
Qilu University of Technology (Shandong Academy of Sciences)
Jinan, China
suntao0906@163.com

Qi Wang

School of Computer Science and Technology
Qilu University of Technology (Shandong Academy of Sciences)
Jinan, China
yining1104@sina.com

Abstract—In complex industrial processes, complex associations are often involved. This complex relationship makes traditional fault detection and diagnosis methods difficult to achieve satisfactory results in multi-source fault detection and diagnosis. Therefore, this paper proposes a new multi-source fault detection and diagnosis framework. The method can successfully detect the state of the system, and at the same time, can locate the fault source simply and quickly. This method firstly constructs the multi-level knowledge graph in the complex industrial process, and then use the discriminant coefficient R to detect whether the system has failed. If the system fails into the fault diagnosis stage, the probability of the fault is derived based on Bayesian theory. This article describes the framework in detail. The TE process is taken as an example to prove the effectiveness of the method.

Keywords—complex industrial process, knowledge graph, bayesian theory, fault detection and diagnosis

I. INTRODUCTION

With the development of computer technology and the promotion of concepts such as Industry 4.0, modern industrial processes tend to be more automated, integrated, complex and intelligent. In the actual process monitoring, due to the large scale of industrial processes and complex business logic, there is a complex relationship between process variables in the production process. At the same time, with the increasing complexity of the process, the influencing factors are gradually increasing, and multiple faults occur frequently in complex industrial systems. However, the traditional fault diagnosis technology is mostly carried out under the condition of single fault type and simple influencing factors, and its accuracy is greatly reduced in the face of complex industrial processes. Many factors make the fault diagnosis of complex industrial system more and more difficult.

How to construct and mine the relationship in complex industrial systems and use these associations to improve the accuracy of fault detection and diagnosis is a key issue that needs to be studied at present. The Knowledge Graph (KG) [1] can describe the various entities and concepts that exist in the real world and the relationships between these entities and concepts. Based on the concept map of knowledge graph, mining the associations in complex industrial systems and displaying them in the form of graphs can more intuitively present this intricate relationship. At the same time, it can describe the abstract concepts of different levels and granularities, and integrate the larger resources in complex industrial systems. The fault source is located by inferring the constructed knowledge graph. Therefore, this paper proposes a new fault detection and diagnosis framework for many

problems in complex industrial systems, which combines knowledge graph and bayesian inference, and conducts experiments on TE data sets to verify its effectiveness.

II. RELATED WORK

Once an accident occurs in a complex industrial process, it may cause serious adverse effects on production safety, efficiency or product quality. At present, the widely used methods in fault detection and diagnosis (FDD) of complex industrial processes include: analytical mathematical model-based method, knowledge-based method, data-driven method, etc.

The analytical model-based approach [2-4] has great limitations in application due to the need to establish accurate mathematical models. Knowledge-based methods are mainly divided into two types: causal graph method [5-7] and fault tree method [8]. Zhang K et al. [5] proposed a kernel-based conditional independence test for conditional independence testing in Bayesian network learning and causal discovery. Experimental results show that it is superior to other methods. Caceres et al. [8] proposed to establish the fault tree according to the structural block diagram of the system. However, the defect of knowledge-based approach is that it relies too much on production experience and process knowledge. The data-driven FDD method [9-10] achieves fault detection and diagnosis of the system by analyzing and processing a large number of process data including system normal and fault information. However, the diagnostic accuracy of data-driven methods often depends heavily on the completeness and representativeness of the fault samples.

Both knowledge-based and data-driven approaches have their own shortcomings. Therefore, this paper proposes a method for fault detection and diagnosis combining knowledge-based methods with data-driven methods. Knowledge graph [1] is an emerging research field in recent years. It can express more information than traditional knowledge-based methods. This method is mainly used in natural language processing, medical[11], financial and other fields. However, the application of this method in complex industrial fields is rare. Knowledge graph is a graph model in essence, and fault diagnosis based on knowledge graph can be solved by graph model theory. However, Bayesian network (BN) is a typical probability graph model, which, considering the network structure and node attribute information, has a solid theoretical basis of probability theory and is widely used. It is suitable for expressing and analyzing uncertain knowledge and can make effective inference to uncertain knowledge. Therefore, this paper uses the combination of knowledge graph and Bayesian theory to detect and diagnose faults in complex industrial systems.

DOI reference number: 10.18293/SEKE2019-064

This work was supported in part by Shandong Natural Resources Fund (NO.02053522), Shandong Province Graduate Education Innovation Program (NO.24170404), and Qilu University of Technology Teaching and Research Project (NO.041201034109).

III. FAULT DETECTION AND DIAGNOSIS METHOD BASED ON MULTI-LEVEL KNOWLEDGE GRAPH

In complex industrial systems, there are many levels of factors that affect the state of production. For example, production level, process level, energy saving and emission reduction level, and raw material level. Therefore, the framework proposed in this paper analyzes complex industrial systems from multiple levels and constructs a multi-level knowledge graph, and then makes inferential diagnosis and fault diagnosis based on this knowledge graph. The flowchart of the framework is shown in Figure 1.

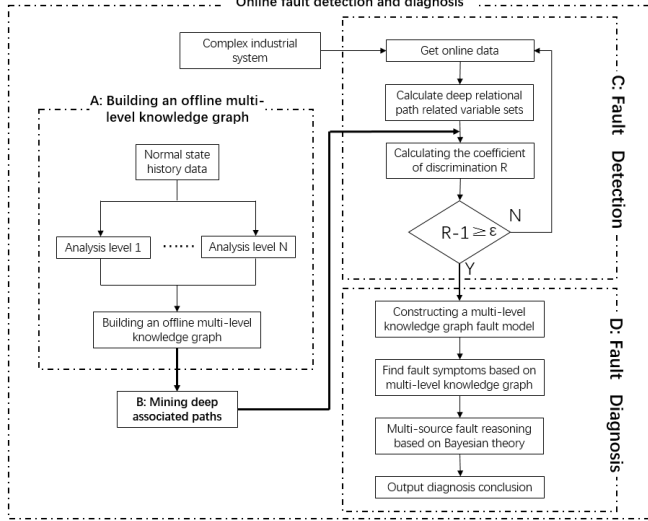


Fig. 1. Schematic diagram of fault detection and diagnosis model based on multi-level knowledge graph

A. Building an offline multi-level knowledge graph

The flow of constructing the knowledge graph in Part A of Figure 1 is shown in Figure 2.

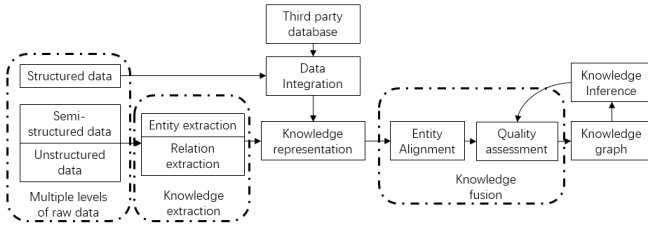


Fig. 2. Schematic diagram of constructing a knowledge graph

There are many differences between the knowledge graph construction of complex industrial systems and the construction of general knowledge graph. Data sources in complex industrial systems come not only from the Internet, but also from all levels affecting production. If the knowledge graph is not comprehensive enough, the time for fault detection and diagnosis will become longer and the accuracy will decrease. First, collect data from multiple levels in a complex industrial system, and then extract the knowledge from the original data. Because the relationship between complex industrial system entities is particularly complex, this paper focuses on the relationship extraction in knowledge extraction. In this paper, Pearson correlation coefficient method is used to find the correlation between entities (the relationship between moderate correlation and high correlation is adopted [12]), and at the same time, prior knowledge is combined to extract the relationship. The extracted knowledge elements are then represented for further

processing. Then there is data fusion, the purpose of which is to fuse the knowledge acquired by different data sources to construct the association between the data. Knowledge graph are constructed by merging knowledge through entity alignment (judging whether two entities match) and quality assessment (inconsistency verification: identifying potential contradictions through rules). Figure 3 is a schematic diagram of the knowledge graph at four levels.

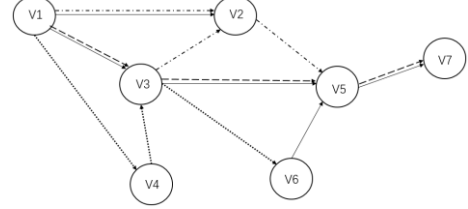


Fig. 3. Schematic diagram of multi-level knowledge graph

B. Mining deep-level associations in multi-level knowledge graphs

The method of mining deep-level associations in multi-level knowledge graphs in Part B of Figure 1 is as follows: According to the existing relationship in the multi-level knowledge graph, find the variable correlation coefficient of each variable in the knowledge graph as the weight coefficient of the variable, and select a critical path according to the size of the weight coefficient. The variables in this path must appear in the knowledge graph of each level at the same time, so that they can represent the deep relationship of the multi-level knowledge graph. Through the deep relationship in the knowledge graph, it is possible to quickly and conveniently detect whether the system has failed without considering the complex relationship in the entire knowledge graph.

C. Fault detection method based on multi-level knowledge graph

The part C fault detection method in Figure 1 is as follows: The knowledge graph is mainly composed of several triples, that is, pairs of variable correlation coefficients. After the system fails, the correlation coefficient of the variables in the original knowledge graph will inevitably change, that is, the entity changes, and the relationship (weight) between the entities changes. Therefore, this paper uses the changes in these two aspects of the knowledge graph structure to define the discriminant coefficient R rule. The discriminant coefficient R rule is defined as follows:

Suppose that T is used to represent the variable in the deep associated path in the knowledge graph under normal conditions, $T=[T_1, T_2, T_3, \dots]$, N_i represents the set of the i-th variable entity pair in T, n represents the number of pairs of entities in N_i . W_i represents the relationship weight of the i-th pair of entities in N. Similarly, T' , N'_i and W'_i are used to represent the variables under test, and m is the number of entity pairs in N'_i . The calculation rules for the discriminant coefficient R are as follows:

Let the number of the variable entity pairs in the normal state T and the variable entity pairs in the state T' to be detected be the same as t, and the set of different entity pairs in the T state and T' is T_k , and the pair of different entities in the T' state and T is T'_k . When T_k , T'_k have different entity pairs, T_k , T'_k are automatically incremented by one. When T_k , T'_k have different entity pairs, W'_i is automatically incremented by 1. When T_k , T'_k have the same entity pair, and W's relationship weight is inconsistent with the corresponding

relationship weight in W' , W' is automatically incremented by one. The mathematical model of the discriminant coefficient R is:

$$R = \frac{\sum T_k + \sum T'_{k+t}}{n} + W' = \frac{|m-t| + |n-t| + t}{n} + W' = \frac{m+n-t}{n} + W' \quad (1)$$

On the basis of an acceptable fault tolerance rate ε , if $R-1 \geq \varepsilon$ is satisfied, it indicates a system failure, otherwise it indicates a normal state.

D. Fault Diagnosis Method Based On Multi-Level Knowledge Graph

The D part fault diagnosis method in Figure 1 is as follows:

1) According to the determined fault variables, the moderately and highly correlated variable pairs of correlation coefficients are selected in the knowledge graph relationship pairs of each level to construct a multi-level knowledge graph fault model.

2) According to the fault symptoms of the system, find all possible candidate failure causes. For each candidate failure reason, the posterior probability value under the known fault symptom condition is calculated based on Bayesian theory.

3) Set a certain threshold, and consider the cause of the failure exceeding the threshold as the most likely cause of the failure.

The posterior probability calculation method is as follows:

1) Calculate the first-order cut set expression for each fault symptom E_i .

$$E_i = e_{t,i}r_t \cup e_{j,i}E_j \quad (2)$$

$$E_j = e_{k,j}r_k \quad (3)$$

Where, E_i and E_j are fault symptoms, r_t and r_k are candidate fault causes, and $e_{j,i}$ is the edge from E_j to E_i . $e_{t,i}$ is the edge from candidate failure cause r_t to failure symptom E_i .

2) Calculate the final cut set expression for each fault symptom E_i :

$$E_i = e_{t,i}r_t \cup e_{j,i}e_{k,j}r_k \quad (4)$$

The first-order cut set expression is expanded according to the relevant edge direction logic. This process can eliminate all the variable nodes in the fault model and obtain the final cut set expression consisting only of the cause node and the relevant edge.

3) Calculate the posterior probability value $P(r_i|E)$ of the fault cause r_i : Logging the fault symptoms according to (5) for all fault symptoms, and inserting the corresponding fault cause occurrence probability value and the associated edge probability value to obtain the posterior probability value $P(r_i|E)$ of the final detected fault cause r_i .

$$P(r_i|E) = \frac{P(r_i E_1 \dots E_n)}{P(E_1 \dots E_n)} = \frac{P(r_i \cap E_1 \dots E_n)}{P(E_1 \cap \dots \cap E_n)} \quad (5)$$

Where, $E = (E = E_1, \dots, E_n)$ is the set of fault symptoms, n is the number of abnormal variables, and r_i is the cause of the fault.

IV. EXPERIMENT

A. Data Sets

The data set used in this paper is the data of TE process, a simulation system of chemical process. For a detailed description of the process, refer to the relevant literature[13]. The 22 process measurement variables in the TE model are represented by V1, V2...V22 and the 20 failure types are r41, r42...r60.

B. Building a multi-level knowledge graph of the TE process

Limited by the TE process simulation data, this paper constructs the TE process knowledge graph from two levels, as shown in Figure 4. The solid line represents the technological process level, and the dotted line represents the data level of the production process.

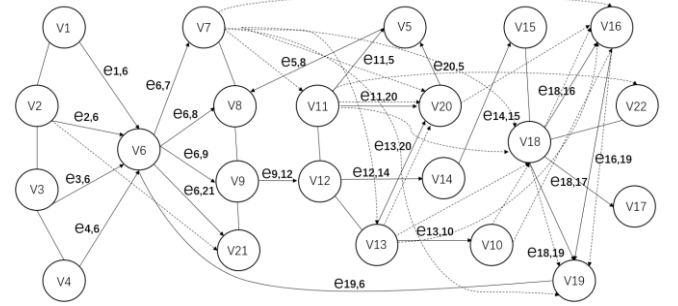


Fig. 4. Schematic diagram of the multi-level knowledge graph of the TE process

C. Mining deep-level associations in multi-level knowledge graph of TE processes

First, the weighting coefficients of each successive measured variable are analyzed according to the constructed multi-level knowledge graph. The weights corresponding to the measured variables $\{V2, V7, V10, V11, V13, V16, V18, V19, V20\}$ are $\{1, 6, 2, 3, 4, 5, 7, 2, 4\}$.

According to the TE process and the principle of large weight coefficient, the deep correlation path of the multi-level knowledge graph can be obtained as follows: V2, V7, V13, V16, V18, V20. Analysis of the four variables of the TE process feed shows that the V2 variable is highly correlated with other variables. This determines that V2 is the starting variable of the deep associated path. This path represents the deep interrelationship of the equipment (feed-reactor-separator-compressor-stripper).

D. TE process fault detection method based on multi-level knowledge graph

In order to prove the validity of the method, this experiment selected single source fault (fault 7), multi-source fault: fault (5, 7) and fault (2, 6, 13).

According to the fault detection method proposed above, the discrimination coefficient R in the four production states is calculated. It is judged whether the TE chemical system is in a fault state according to the change of the discrimination coefficient R . The parameters R in the four production states are shown in Figure 5. It can be seen from Figure 5 that the discriminant coefficient R completely separates the data of the normal state and the fault state, and accurately detects the system fault.

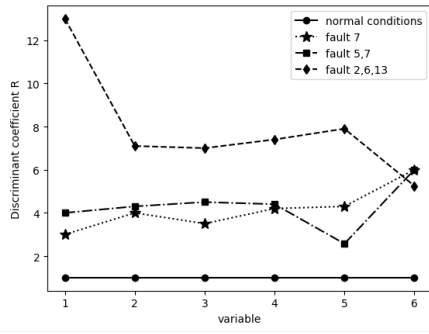


Fig. 5. Schematic diagram of the comparison between the normal state and the fault state discrimination coefficient R

E. TE process fault diagnosis method based on multi-level knowledge graph

In this experiment, fault 5 and fault 7 were selected as the multi-source fault data set for experiment. The specific diagnostic steps are as follows:

1) Building a knowledge graph failure model in the current state as shown in Figure 6.

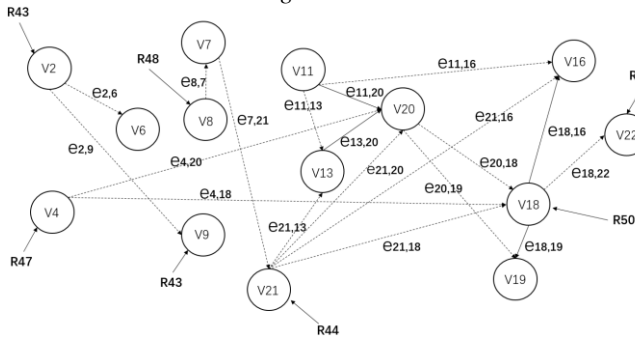


Fig. 6. Schematic diagram of multi-level knowledge graph failure model

2) Calculate the posterior probability value of each candidate failure cause.

According to (2)-(5), the diagnosis results are shown in Table 1. It can be seen from the table that the posterior probability of r_{45} and r_{47} is significantly larger than r_{43} , r_{44} , r_{48} , r_{50} . Therefore, the cause node r_{45} (condenser cooling water feed temperature) and r_{47} (flow rate 4 C pressure loss) are the cause of the failure, and the diagnosis conclusion is consistent with the original assumption.

TABLE I. DIAGNOSTIC RESULTS OF THE CAUSE NODE

Candidate failure cause	Percentage of posterior probability
r43:D temperature in flow 2	14.11%
r44: Reactor cooling water feed temperature	7.29%
r45: condenser cooling water feed temperature	65.47%
r47: C pressure loss in flow 4	50.32%
r48: Changes in A, B, and C components in Flow 4	10.51%
r50: C material temperature in flow 4	25.87%

TABLE II. DIAGNOSTIC TIME COMPARISON

Diagnosis method	Diagnostic time
PCA_KNN	0.446
PCA_SVM	0.413
PNN	0.375
KG_ Bayesian network	0.306

Table 2 compares the diagnostic time of the four methods of PCA_KNN, PCA_SVM, PNN, KG_ Bayesian network. It can be seen that the method used in this paper takes less time than the traditional fault detection and diagnosis methods.

V. CONCLUSION

Based on the strong correlation between knowledge graphs and comprehensive consideration of various factors, this paper constructs a multi-level knowledge graph in complex industrial processes. This way of building knowledge graphs maximizes the consideration of influencing factors in complex industrial systems. By mining the correlation law between variables, the discriminant coefficient method is used to detect the state of the system. The method has achieved good results in the TE process and can accurately identify the system state. The multi-level knowledge graph fault diagnosis method based on Bayesian theory provides a new idea for fault diagnosis of complex industrial systems. This method has achieved good results in reasoning the multi-source failure of complex industrial systems. At the same time, this method can diagnose the fault source easily and quickly. At present, this method is only applicable to the processing of existing fault cause data, and the next step will be to study the fault self-learning function.

REFERENCES

- [1] Huang Z , Chung W , Ong T H , et al. [ACM Press the second ACM/IEEE-CS joint conference - Portland, Oregon, USA (2002.07.14-2002.07.18)] Proceedings of the second ACM/IEEE-CS joint conference on Digital libraries, - JCDL '02 - A graph-based recommender system for digital library[J]. 2002:65
- [2] Kinnaert, Michel. Fault diagnosis based on analytical models for linear and nonlinear systems - a tutorial[J]. IFAC Proceedings Volumes, 2003, 36(5):37-50.
- [3] Liao Z , Wen F , Guo W , et al. An analytic model and optimization technique based methods for fault diagnosis in power systems[C]// International Conference on Electric Utility Deregulation & Restructuring & Power Technologies. IEEE, 2008.
- [4] Cui Y , Shi J , Wang Z . An analytical model of electronic fault diagnosis on extension of the dependency theory[J]. Reliability Engineering & System Safety, 2015, 133:192-202.
- [5] Zhang K, Peters J, Janzing D, et al. Kernel-based conditional independence test and application in causal discovery[J]. arXiv preprint arXiv:1202.3775, 2012
- [6] Iri M , Aoki K , O'Shima E , et al. An algorithm for diagnosis of system failures in the chemical process[J]. Computers & Chemical Engineering, 1979, 3(1-4):489-493
- [7] Yang F , Sirish L S , Xiao D . Signed Directed Graph modeling of industrial processes and their validation by data-based methods[C]// Control & Fault-tolerant Systems. IEEE, 2010
- [8] Caceres S , Henley E J . Process Failure Analysis by Block Diagrams and Fault Trees[J]. Industrial & Engineering Chemistry Research, 1976, 15(2):128-134
- [9] Zhang J , Zhu Y , Shi W , et al. An Improved Machine Learning Scheme for Data-Driven Fault Diagnosis of Power Grid Equipment[C]// IEEE International Conference on High Performance Computing & Communications. IEEE, 2015
- [10] Wang D , Man Z . Special issue: Data-driven fault diagnosis of industrial systems[J]. Information Sciences, 2014, 259:231-233
- [11] Fang Y , Wang H , Wang L , et al. Diagnosis of COPD Based on a Knowledge Graph and Integrated Model[J]. IEEE Access, 2019, 7: 46004-46013
- [12] Kim Y , Kim T H , Ergün T . The instability of the Pearson correlation coefficient in the presence of coincidental outliers[J]. Finance Research Letters, 2015, 13: 243-257
- [13] Chiang L H , Russell E L , Braatz R D . Fault detection and diagnosis in industrial systems[J]. 2001.

Formal Specification and Model Checking of A* Algorithm

Kazuhiro Ogata

School of Information Science

Japan Advanced Institute of Science and Technology (JAIST)

1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

Email: ogata@jaist.ac.jp

Abstract—A* algorithm is formally specified in Maude and model checked with the Maude LTL model checker. We take into account a graph such that it is a DAG, a goal node is reachable from a start node and each edge is given a non-negative weight. If h is admissible, namely that $h(n)$ never overestimates the cost to the goal from n for all nodes n , then A* finds a shortest path. The condition, however, can be relaxed. Our model checking experiments make us conjecture that if there exists a shortest path such that for each node n in the path $h(n)$ plus the cost to n from the start node is less than the cost of any non-shortest path to the goal from the start, A* finds a shortest path.

Keywords—A* algorithm; Dijkstra algorithm; LTL; Maude; Model checking

I. INTRODUCTION

A* algorithm[1] is a generalized version of Dijkstra (shortest path finding) algorithm[2]. It uses an estimation of the distance between the goal node and each of the edge nodes to select the next node to be tackled. Dijkstra algorithm treats the estimation as 0.

A* is formally specified in Maude[3], a rewriting logic-based specification/programming language equipped with many facilities, among which are model checking ones (a reachability analyzer and an LTL model checker). We model check with the LTL model checker that A* terminates and finds a shortest path. Our model checking experiments say that A* always halts and if the estimation h is admissible, namely that $h(n)$ never overestimates the cost to the goal from n for all nodes n , then A* finds a shortest path. The condition, however, can be relaxed. Our model checking experiments make us conjecture that if there exists a shortest path such that for each node n in the path $h(n)$ plus the cost to n from the start node is less than the cost of any non-shortest path to the goal from the start, A* finds a shortest path.

The contribution of the work described in the present paper is to demonstrate how A* is formally specified in Maude and model checked with the Maude LTL model checker and to find a relaxed sufficient condition, through model checking experiments, that A* finds a shortest path.

This work was partially supported by JSPS KAKENHI Grant Number JP26240008 & JP19H04082.

DOI reference number: 10.18293/SEKE2019-022

Saberi, Groote and Keshishzadeh [4] have formally specified in the mCRL2 language a simple path planning algorithm that makes multiple robots reach a destination on a discretized planar surface, described properties in μ -calculus and conducted model checking experiments that the algorithm enjoys the properties. The desired properties they have taken into account are deadlock-freeness, collision-freeness and reachability. Their motivation to conduct the research is to demonstrate how useful model checking is to verify that the collective behavior of multiple robots satisfy some desired properties. Maragos, Kleftouris and Ziogou [5] have formally modeled a path finding planning in Colored Petri Nets (CP-Nets). Given a discretized planar space where there is one robot located in one position, there are some obstacles located in some positions and there is a goal position, the planning is to find a path along which the robot reaches the goal. Their CP-Nets model is executable with the Design/CPN tool. Their main motivation to conduct the research is to demonstrate how powerful CP-Nets are for simulating such a planning. Alphan, Smith, Belta and Rus [6] have proposed a method that automatically plans optimal paths for a group of robots that satisfy LTL properties. They formally modeled as a timed automaton a system in which multiple robots asynchronously move to some vertexes from the current vertexes on a graph if there are direct edges from the latter to the former. They use a bi-simulation relation between a timed automaton that is inherently infinite and a finite-state transition system so that they can use their earlier algorithm dedicated to a single-robot system.

The rest of the paper is organized as follows: § II Preliminaries, § III A* Algorithm, § IV Formal Specification, § V Model Checking, and § VI Conclusion.

II. PRELIMINARIES

A Kripke structure K is $\langle S, I, T, P, L \rangle$, where S is a set of states, $I \subseteq S$ is the set of initial states, $T \subseteq S \times S$ is a total binary relation over S , P is a set of atomic propositions and L is a labeling function whose type is $S \rightarrow 2^P$. Each element $(s, s') \in T$ is called a state transition from s to s' and T may be called the state transitions (with respect to K). For a state $s \in S$, $L(s)$ is the set of atomic propositions that hold in s . A path π is an infinite sequence $s_0, \dots, s_i, s_{i+1}, \dots$ of states such that $s_i \in S$ and

$(s_i, s_{i+1}) \in T$ for each i . Let π^i be s_i, s_{i+1}, \dots and $\pi(i)$ be s_i . Let P be the set of all paths. π is called a computation if $\pi(0) \in I$. Let C be the set of all computations.

The syntax of a formula φ in LTL for K is $\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi$, where $p \in P$. Let \mathcal{F} be the set of all formulas in LTL for K . An arbitrary path $\pi \in \mathcal{P}$ of K and an arbitrary LTL formula $\varphi \in \mathcal{F}$ of K , $K, \pi \models \varphi$ is inductively defined as $K, \pi \models \top$, $K, \pi \models p$ iff $p \in L(\pi(0))$, $K, \pi \models \neg\varphi_1$ iff $K, \pi \not\models \varphi_1$, $K, \pi \models \varphi_1 \wedge \varphi_2$ iff $K, \pi \models \varphi_1$ and $K, \pi \models \varphi_2$, $K, \pi \models \bigcirc \varphi_1$ iff $K, \pi^1 \models \varphi_1$, and $K, \pi \models \varphi_1 \mathcal{U} \varphi_2$ iff there exists a natural number $j < i$, $K, \pi^j \models \varphi_2$ and for all natural numbers $j < i$, $K, \pi^j \models \varphi_1$, where φ_1 and φ_2 are LTL formulas. Then, $K \models \varphi$ iff $K, \pi \models \varphi$ for each computation $\pi \in \mathcal{C}$ of K . The temporal connectives \bigcirc and \mathcal{U} are called the next connective and the until connective, respectively. The other logical and temporal connectives are defined as usual as follows: $\perp \triangleq \neg\top$, $\varphi_1 \vee \varphi_2 \triangleq \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \Rightarrow \varphi_2 \triangleq \neg\varphi_1 \vee \varphi_2$, $\Diamond \varphi \triangleq \top \mathcal{U} \varphi$, and $\Box \varphi \triangleq \neg(\Diamond \neg\varphi)$. The temporal connectives \Diamond and \Box are called the eventually connective and the always connective, respectively.

There are multiple possible ways to express states. We express a state as a braced associative-commutative (AC) collection of name-value pairs. AC collections are called soups, and name-value pairs are called observable components. That is, a state is expressed as a braced soup of observable components. The juxtaposition operator is used as the constructor of soups. Let oc_1, oc_2, oc_3 be observable components, and then $oc_1 \ oc_2 \ oc_3$ is the soup of those three observable components. A state is expressed as $\{oc_1 \ oc_2 \ oc_3\}$. There are multiple possible ways to specify state transitions. We specify them as rewrite rules. Concretely, we use Maude [3], a programming/specification language based on rewriting logic. Maude makes it possible to specify complex systems flexibly and is also equipped with model checking facilities (a reachability analyzer and an LTL model checker). A conditional rewrite rule (or just a rule) is in the form $\text{crl } [lb] : l \Rightarrow r \text{ if } \dots \wedge c_i \wedge \dots$, where lb is the label given to the rule and c_i is part of the condition, which may be an equation $lc_i = rc_i$. The negation of $lc_i = rc_i$ could be written as $(lc_i \neq rc_i) = \text{true}$, where $= \text{true}$ could be omitted. If the condition $\dots \wedge c_i \wedge \dots$ holds under some substitution σ , $\sigma(l)$ can be replaced with $\sigma(r)$.

The search command tries to find a state reachable from t such that the state matches p and satisfies c :

```
search [1] in M : t => * p such that c .
```

where M is a specification of the S and T parts of K . t typically represents an initial state of K .

Let $init$ be the only initial state of K and φ be an LTL formula. Then, the Maude LTL model checker checks that K satisfies φ by reducing `modelCheck (init, φ)`.

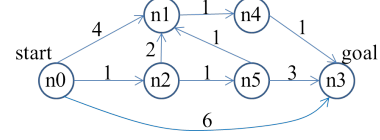


Figure 1. A DAG (1)

III. A* ALGORITHM

Let us consider the DAG shown in Fig. 1 in which node $n0$ is the start node and node $n3$ is the goal node. Let us use the estimation $h(n)$ for each node n defined as follows: $h(n1) = 2$, $h(n3) = 0$ and $h(n) = 1$ for any other node n . Let $w(n, n')$ be the weight of the edge from node n to node n' . For example, $w(n2, n1) = 2$. For each node n , two pieces $p(n)$ and $d(n)$ of information are maintained, where $p(n)$ is the shortest path found so far to node n from the start node at any given moment and $d(n)$ is its distance. One more piece oq of information is maintained, which is an ordered queue of node & natural number pairs, where such pairs are stored in increasing order based on their second elements. When a pair $\langle n, d \rangle$ of node n and natural number d is enqueued into oq that contains some other pairs whose second elements are d , $\langle n, d \rangle$ is stored following all those pairs. When $\langle n, d \rangle$ is enqueued into oq that contains $\langle n, d' \rangle$, if $d' < d$, then $\langle n, d \rangle$ is stored at the designated place in oq and $\langle n, d' \rangle$ is deleted and otherwise $\langle n, d \rangle$ is not stored in oq . For example, let oq consist of $\langle n0, 1 \rangle$, $\langle n1, 1 \rangle$, $\langle n3, 2 \rangle$ and $\langle n2, 3 \rangle$ in this order, denoted $\langle n0, 1 \rangle \mid \langle n1, 1 \rangle \mid \langle n3, 2 \rangle \mid \langle n2, 3 \rangle$, and if $\langle n3, 1 \rangle$ is put into oq , oq is $\langle n0, 1 \rangle \mid \langle n1, 1 \rangle \mid \langle n3, 1 \rangle \mid \langle n2, 3 \rangle$. We use `empq` to represent the empty queue and suppose that `empq` is an identity of the queue constructor \mid , namely that $q \mid \text{empq} = q$ and $\text{empq} \mid q = q$ for all queues q . We also suppose that a single element e is treated as a singleton queue that only consists of e . For example, $\langle n3, 1 \rangle$ is treated as the singleton ordered queue that only consists of $\langle n3, 1 \rangle$. Node n of each $\langle n, d \rangle$ in oq at any moment is called an edge node at that moment in this paper.

Initially, $p(n0) = n0$ (which is the path that only consists of $n0$), $p(n) = \varepsilon$ (the empty path) for any other node n , $d(n0) = 0$, $d(n) = \infty$ for any other node n and $oq = \langle n0, 1 \rangle$ (which is the ordered queue that only consists of $\langle n0, 1 \rangle$, where $d(n0) + h(n0) = 1$).

While oq is not empty, the following is repeated. Let $\langle n, d \rangle$ be the top element of oq . If node n is the goal node, $p(n)$ is the path to be found from the start node to the goal node. Otherwise, for each direct successor node n' of node n , if $d(n) + w(n, n') < d(n')$, $d(n')$ is set to $d(n) + w(n, n')$, $p(n')$ is set to the path obtained by adding n' to $p(n)$ at the end and $\langle n', d(n) + w(n, n') + h(n') \rangle$ is enqueued into oq . The top element is deleted from oq .

At some moment,

- $p(n0) = n0 \ \& \ d(n0) = 0$
- $p(n1) = (n0 \rightarrow n1) \ \& \ d(n1) = 4$
- $p(n2) = (n0 \rightarrow n2) \ \& \ d(n2) = 1$
- $p(n3) = (n0 \rightarrow n3) \ \& \ d(n3) = 6$
- $p(n) = \varepsilon \ \& \ d(n) = \infty$ for any other node n
- $oq = \langle n2, 2 \rangle \mid \langle n1, 6 \rangle \mid \langle n3, 6 \rangle$

$\langle n2, 2 \rangle$ is the top element of oq . $n1$ and $n5$ are the direct successor nodes of $n2$. Since $d(n2) + w(n2, n1) = 3$, $d(n1) = 4$ and $3 < 4$, $d(n1)$ is set to 3, $p(n1)$ is set to $(n0 \rightarrow n2 \rightarrow n1)$ and $\langle n1, 5 \rangle$, where $d(n2) + w(n2, n1) + h(n1) = 5$, is enqueued into oq . Since $d(n2) + w(n2, n5) = 2$, $d(n5) = \infty$ and $2 < \infty$, $d(n5)$ is set to 2, $p(n5)$ is set to $(n0 \rightarrow n2 \rightarrow n5)$ and $\langle n5, 3 \rangle$, where $d(n2) + w(n2, n5) + h(n5) = 3$, is enqueued into oq . The top element is deleted from oq . At this moment,

- $p(n0) = n0 \ \& \ d(n0) = 0$
- $p(n1) = (n0 \rightarrow n2 \rightarrow n1) \ \& \ d(n1) = 3$
- $p(n2) = (n0 \rightarrow n2) \ \& \ d(n2) = 1$
- $p(n3) = (n0 \rightarrow n3) \ \& \ d(n3) = 6$
- $p(n4) = \varepsilon \ \& \ d(n4) = \infty$
- $p(n5) = (n0 \rightarrow n2 \rightarrow n5) \ \& \ d(n5) = 2$
- $oq = \langle n5, 3 \rangle \mid \langle n1, 5 \rangle \mid \langle n3, 6 \rangle$

$\langle n5, 3 \rangle$ is the top element of oq . $n1$ and $n3$ are the direct successor nodes of $n5$. Since $d(n5) + w(n5, n1) = 3$, $d(n1) = 3$ and $3 \not< 3$, nothing changes. Since $d(n5) + w(n5, n3) = 5$, $d(n3) = 6$ and $5 < 6$, $d(n3)$ is set to 5, $p(n3)$ is set to $(n0 \rightarrow n2 \rightarrow n5 \rightarrow n3)$ and $\langle n3, 5 \rangle$, where $d(n5) + w(n5, n3) + h(n3) = 5$, is enqueued into oq . The top element is deleted from oq . At this moment,

- $p(n0) = n0 \ \& \ d(n0) = 0$
- $p(n1) = (n0 \rightarrow n2 \rightarrow n1) \ \& \ d(n1) = 3$
- $p(n2) = (n0 \rightarrow n2) \ \& \ d(n2) = 1$
- $p(n3) = (n0 \rightarrow n2 \rightarrow n5 \rightarrow n3) \ \& \ d(n3) = 5$
- $p(n4) = \varepsilon \ \& \ d(n4) = \infty$
- $p(n5) = (n0 \rightarrow n2 \rightarrow n5) \ \& \ d(n5) = 2$
- $oq = \langle n1, 5 \rangle \mid \langle n3, 5 \rangle$

$\langle n1, 5 \rangle$ is the top element of oq . $n4$ is the direct successor node of $n1$. Since $d(n1) + w(n1, n4) = 4$, $d(n4) = \infty$ and $4 < \infty$, $d(n4)$ is set to 4, $p(n4)$ is set to $(n0 \rightarrow n2 \rightarrow n1 \rightarrow n4)$ and $\langle n4, 5 \rangle$, where $d(n1) + w(n1, n4) + h(n4) = 5$, is enqueued into oq . The top element is deleted from oq . At this moment,

- $p(n0) = n0 \ \& \ d(n0) = 0$
- $p(n1) = (n0 \rightarrow n2 \rightarrow n1) \ \& \ d(n1) = 3$
- $p(n2) = (n0 \rightarrow n2) \ \& \ d(n2) = 1$
- $p(n3) = (n0 \rightarrow n2 \rightarrow n5 \rightarrow n3) \ \& \ d(n3) = 5$
- $p(n4) = (n0 \rightarrow n2 \rightarrow n1 \rightarrow n4) \ \& \ d(n4) = 4$
- $p(n5) = (n0 \rightarrow n2 \rightarrow n5) \ \& \ d(n5) = 2$
- $oq = \langle n3, 5 \rangle \mid \langle n4, 5 \rangle$

$\langle n3, 5 \rangle$ is the top element of oq . Since $n3$ is the goal node, we have found the path $n0 \rightarrow n2 \rightarrow n5 \rightarrow n3$ whose distance (or cost) is 5. The path is one of the three shortest paths from $n0$ to $n3$ in DAG (1) shown in Fig. 1.

In this paper, we take into account a graph such that it is a DAG, a goal node is reachable from a start node and each edge is given a non-negative weight.

IV. FORMAL SPECIFICATION

Let K_{A^*} be the Kripke structure formalizing A^* that tackles a DAG in which there are N nodes.

The four kinds of observable components are used:

- $(node[ni] : d, ps_1, ps_2, p) - ni$ is a node ID, d is a natural number or ∞ that is the distance of the path found so far from a start node to the node ni , ps_1 & ps_2 are soups of the direct successor node IDs of the node ni that have not yet been tackled & that have already been tackled and p is the path found so far from a start node to the node ni ; if d is ∞ , no path from a start node to the node ni has yet been found;
- $(oq : q) - q$ is an ordered queue of node ID & natural number pairs;
- $(path : npp) - npp$ is a pair of a natural number and a path; when a path from a start node to a goal node is found, the pair of the distance of the path and the path is stored in it;
- $(gstat : gs) - gs$ is either $nFin$ or Fin ; a path from a start node to a goal node has been found if gs is Fin .

Each state in S_{A^*} is expressed as $\{obs\}$, where obs is a soup of those observable components such that there is one $gstat$ observable component, there is one $path$ observable component, there is one oq observable component and there are N node observable components.

I_{A^*} consists of one state. When DAG (1) shown in Fig. 1 is tackled, the initial state is expressed as follows:

```
{(gstat: nFin)
(oq: (< n0, 0 >)) (path: (< 0, nil >))
(node[n0]: 0, (< n1, 4 > < n2, 1 > < n3, 6 >),
empty, n0)
(node[n1]: oo, (< n4, 1 >), empty, nil)
(node[n2]: oo, (< n1, 2 > < n5, 1 >), empty, nil)
(node[n3]: oo, empty, empty, nil)
(node[n4]: oo, (< n3, 1 >), empty, nil)
(node[n5]: oo, (< n1, 1 > < n3, 3 >), empty, nil)}
```

where $\langle n0, 0 \rangle$ in the oq observable component represents the singleton ordered queue that only consists of $\langle n0, 0 \rangle$, nil is the empty path, $empty$ is the empty soup, $\langle n1, 4 \rangle \mid \langle n2, 1 \rangle \mid \langle n3, 6 \rangle$ is the soup that consists of $\langle n1, 4 \rangle$, $\langle n2, 1 \rangle$ and $\langle n3, 6 \rangle$, and oo is ∞ .

T_{A^*} is specified as six rewrite rules. Let OCs be a Maude variable of observable component soups, NI & NI' be Maude variables of node IDs, D , W , D'' & D''' be Maude variables of natural numbers, D' be a Maude variable of natural numbers or ∞ , Q be a Maude variable of ordered queues, $NNPs1$, $NNPs2$, $NNPs1'$ & $NNPs2'$ be Maude variables of node ID & natural number pair soups, and L , L' & L'' be Maude variables of paths.

The first one is as follows:

```
rl [A*-stutter] : {(gstat: fin) OCs}
=> {(gstat: fin) OCs} .
```

We need to use rule `stutter` to make T_{A^*} total.

The second one is as follows:

```
cr1 [A*-goal] :
{(gstat: nFin) (oq: (< NI,D''' > | Q))
 (path: (< D'',L'' >))
 (node[NI]: D,NNPs1,NNPs2,L) OCs}
=>
{(gstat: fin) (oq: (< NI,D''' > | Q))
 (path: (< D,L >))
 (node[NI]: D,NNPs1,NNPs2,L) OCs}
if goal?(NI) .
```

`goal?(NI)` holds if `NI` is the goal node ID. When the first element `NI` of the top element $\langle NI, D''' \rangle$ of the ordered queue stored in the `oq` observable component is the goal node ID, then the path `L` has been found and $\langle D, L \rangle$, where `D` is its cost, is stored in the path observable component.

The third one is as follows:

```
cr1 [A*-srch1] :
{(gstat: nFin) (oq: (< NI,D'' > | Q))
 (node[NI]: D,< NI',W > NNPs1,NNPs2,L)
 (node[NI']: D',NNPs1',NNPs2',L') OCs}
=>
{(gstat: nFin)
 (oq: enq(< NI,D'' > | Q,
          < NI',(D + W + h(NI')) >))
 (node[NI]: D,NNPs1,< NI',W > NNPs2,L)
 (node[NI']: D + W,NNPs1',NNPs2',L -> NI')
 OCs}
if D + W < D' /\ NNPs1 != empty .
```

Rule `srch1` says that if the `gstate` observable component is `nFin` (meaning that the path has not been found), `NI` is the node ID found in the top element pair of the ordered queue stored in the `oq` observable component, `NI'` is a direct successor node ID such that it has not been tackled and the weight of the edge between `NI` and `NI'` is `W`, `D + W` is less than `D'` and `NNPs1` is not empty, then $\langle NI', (D + W + h(NI')) \rangle$ is put into the ordered queue stored in the `oq` observable component, $\langle NI', W \rangle$ is moved to the second soup from the first soup in the `node[NI]` observable component (meaning that `NI'` has been tackled), the path to `NI'` from the start node is updated as `L -> NI'` obtained by adding `NI'` to `L` at the end and the distance of the path is updated as `D + W`.

The fourth one is as follows:

```
cr1 [A*-srch2] :
{(gstat: nFin) (oq: (< NI,D'' > | Q))
 (node[NI]: D,< NI',W > NNPs1,NNPs2,L)
 (node[NI']: D',NNPs1',NNPs2',L') OCs}
=>
{(gstat: nFin)
 (oq: enq(Q,< NI',(D + W + h(NI')) >))
```

```
(node[NI]: D,NNPs1,< NI',W > NNPs2,L)
(node[NI']: D + W,NNPs1',NNPs2',L -> NI')
OCs}
if D + W < D' /\ NNPs1 = empty .
```

The only difference between the situations dealt with by rule `srch1` and rule `srch2` is whether `NNPs1` is empty. If `NNPs1` is empty, the top element pair $\langle NI, D'' \rangle$ is deleted from the ordered queue stored in the `oq` observable component.

The fifth one is as follows:

```
cr1 [A*-srch3] :
{(gstat: nFin) (oq: (< NI,D'' > | Q))
 (node[NI]: D,< NI',W > NNPs1,NNPs2,L)
 (node[NI']: D',NNPs1',NNPs2',L') OCs}
=>
{(gstat: nFin) (oq: (< NI,D'' > | Q))
 (node[NI]: D,NNPs1,< NI',W > NNPs2,L)
 (node[NI']: D',NNPs1',NNPs2',L') OCs}
if not (D + W < D') /\ NNPs1 != empty .
```

The only difference between the situations dealt with by rule `srch1` and rule `srch3` is whether `D + W < D'` holds. If `D + W < D'` does not, nothing changes except that $\langle NI', W \rangle$ is moved to the second soup from the first soup in the `node[NI]` observable component.

The sixth one `A*-srch4` deals with the case in which `D + W < D'` does not hold and `NNPs1` is empty. If so, $\langle NI', W \rangle$ is moved to the second soup from the first soup in the `node[NI]` observable component and the top element pair $\langle NI, D'' \rangle$ is deleted from the ordered queue stored in the `oq` observable component.

V. MODEL CHECKING

Let us consider a straightforward algorithm to find all paths from a start node to a goal node for a given graph. The algorithm is formalized as part of K_{all} because we do not need to use P_{all} and L_{all} .

The two kinds of observable components are used:

- $(node[ni]: nstat, n, ps_1, ps_2, ps) - ni$ is a node ID, $nstat$ is a node status (which is one of `notYet`, `visited` and `done`), n is the number of the incoming edges that have not been tackled, ps_1 & ps_2 are soups of the direct successor node IDs of the node ni that have not yet been tackled & that have already been tackled and ps is a soup of (d, p) -pairs, where p is a path from a start node to this node that has been found so far and d is the distance; if $nstat$ is `notYet`, the node has not been tackled, if it is `visited`, the node has been visited (partially tackled) and if it is `done`, the node has been fully tackled;
- $(gstat: gs) - gs$ is either `nFin` or `Fin`; all paths from a start node to a goal node have been found if gs is `Fin`.

I_{all} consists of one state. When DAG (1) shown in Fig. 1 is tackled, the initial state is expressed as follows:

```
{(gstat2: nFin)
(node[n0]: done,0,(< n1,4 > < n2,1 > < n3,6 >),
  empty,< 0,n0 >)
(node[n1]: notYet,3,(< n4,1 >),empty,empty)
(node[n2]: notYet,1,(< n1,2 > < n5,1 >),empty,
  empty)
(node[n3]: notYet,3,empty,empty,empty)
(node[n4]: notYet,1,(< n3,1 >),empty,empty)
(node[n5]: notYet,1,(< n1,1 > < n3,3 >),empty,
  empty)}
```

T_{all} is specified as five rewrite rules. In addition to the Maude variables above-mentioned, let NLs & NLs' be Maude variables of soups of (natural number,path)-pairs, N' is a Maude variable of natural numbers and NS' is a Maude variable of node statuses. The first rule $All-stutter$ is essentially the same as the one of T_{A*} . The second one is as follows:

```
cr1 [All-done] : {(gstat2: nFin)
  (node[NI]: done,0,NNPs1,NNPs2,NLs) OCs}
=> {(gstat2: fin)
  (node[NI]: done,0,NNPs1,NNPs2,NLs) OCs}
if goal?(NI) .
```

The rule says that if all paths to the goal node from the start node have been found, the value of the $gstat2$ observable component is set to fin .

The last three rules are as follows:

```
rl [All-srch1] :
  {(gstat2: nFin)
  (node[NI]: visited,0,NNPs1,NNPs2,NLs) OCs}
=>
  {(gstat2: nFin)
  (node[NI]: done,0,NNPs1,NNPs2,NLs) OCs} .

rl [All-srch2] :
  {(gstat2: nFin)
  (node[NI]: done,0,< NI',W > NNPs1,NNPs2,NLs)
  (node[NI']: notYet,s(N'),NNPs1',NNPs2',NLs') OCs}
=>
  {(gstat2: nFin)
  (node[NI]: done,0,NNPs1,< NI',W > NNPs2,NLs)
  (node[NI']: visited,N',NNPs1',NNPs2',
    add(NLs,NI',W)) OCs} .

cr1 [All-srch3] :
  {(gstat2: nFin)
  (node[NI]: done,0,< NI',W > NNPs1,NNPs2,NLs)
  (node[NI']: NS',s(N'),NNPs1',NNPs2',NLs') OCs}
=>
  {(gstat2: nFin)
  (node[NI]: done,0,NNPs1,< NI',W > NNPs2,NLs)
  (node[NI']: NS',N',NNPs1',NNPs2',
    NLs' add(NLs,NI',W)) OCs}
if NS' /= notYet .
```

where s of $s(N')$ is the successor function of natural numbers.

If the first and second values of the node $[NI]$ observable component are $visited$ and 0 , the node NI has been fully tackled. If that is the case, the rule $All-srch1$ changes the first value into $done$.

If the node NI has been fully tackled and has a direct successor NI' that has not been treated and the first value of the node $[NI']$ observable value is $notYet$, the first

and second values of the node $[NI']$ observable value are changed into $visited$ and the number N' obtained by decrementing the original value $s(N')$; moreover, NI' and W (which is the weight of the edge from NI to NI') are added to each path in the soup NLs of the node $[NI']$ observable value. This is done by the rule $All-srch2$.

The rule $All-srch3$ does almost the same thing as the rule $All-srch2$. The difference is that the rule $All-srch3$ deals with the case where the first value of the node $[NI']$ observable value is not $notYet$, while the rule $All-srch2$ deals with the case where it is $notYet$.

The search command, where $ALLPF$ is the specification of the straightforward algorithm and $init$ is the initial state for $ALLPF$ and the DAG shown in Fig 1,

```
search [1] in ALLPF : init
=>* {(node[n3]: done,0,empty,NNPs2,NLs) OCs} .
```

finds all paths to the node $n3$ from the node $n0$, which are assigned to NLs :

```
NLs --> < 5,n0 -> n2 -> n5 -> n3 >
< 5,n0 -> n2 -> n1 -> n4 -> n3 >
< 5,n0 -> n2 -> n5 -> n1 -> n4 -> n3 >
< 6,n0 -> n3 > < 6,n0 -> n1 -> n4 -> n3 >
```

among which there are three shortest paths:

```
< 5,n0 -> n2 -> n5 -> n3 >
< 5,n0 -> n2 -> n1 -> n4 -> n3 >
< 5,n0 -> n2 -> n5 -> n1 -> n4 -> n3 >
```

Let $sPaths$ refer to the soup of the three shortest paths. The extraction of shortest paths from all paths to a goal node from a start node is done by a program written in Maude.

To model check that K_{A*} satisfies some desired properties, we define (or specify) P_{A*} and L_{A*} . P_{A*} has two atomic propositions: fin and $isSPath$. L_{A*} is specified as follows:

```
eq {(gstat: fin) OCs} |= fin = true .
eq {(path: (< D,L >)) OCs} |= isSPath
  = (< D,L > \in sPaths) .
eq {OCs} |= PROP = false [otherwise] .
```

The three equations say that fin holds a state s iff s contains $(gstat: fin)$ and $isSPath$ holds for a state s iff s contains $(path: (< D,L >))$ and $sPaths$ contains $< D,L >$ as an element. Let two LTL formulas $halt$ and $correct$ be defined as $<> fin$ and $[] (fin -> <> isSPath)$, where $<>$ is \Diamond , $[]$ is \Box and $->$ is \Rightarrow .

Let h be as follows: $h(n1) = 2$, $h(n3) = 0$ and $h(n) = 1$ for any other node n , and $init$ be the initial state for the specification $ASTAR$ of K_{A*} and the DAG shown in Fig 1. We model check that K_{A*} satisfies $halt$ and $correct$ as follows:

```
red modelCheck(init, halt) .
red modelCheck(init, correct) .
```

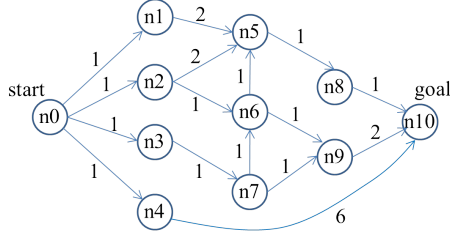


Figure 2. A DAG (2)

No counterexample is found for both model checking experiments. Because h is admissible, namely that it never overestimates the actual cost to the goal $n3$ from any node, K_{A^*} satisfies `correct`.

Let us use another h : $h(n1) = 3$, $h(n5) = 4$, $h(n3) = 0$ and $h(n) = 1$ for any other node n . This h is not admissible because the actual cost to the goal $n3$ from the node $n1$ is 2 and the actual cost to the goal $n3$ from the node $n5$ is 3. The first model checking experiment does not find any counterexamples, while the second one does. This is because h is not admissible. For this case, K_{A^*} finds $n0 \rightarrow n3$, which is not the shortest one.

Let us use yet another h : $h(n1) = 3$, $h(n3) = 0$ and $h(n) = 1$ for any other node n . This h is not admissible either because the actual cost to the goal $n3$ from the node $n1$ is 2. The two model checking experiments, however, do not find any counterexamples, meaning that K_{A^*} still satisfies `correct`. In this case, K_{A^*} finds $n0 \rightarrow n2 \rightarrow n5 \rightarrow n3$. This is because for each node n in the path, $h(n)$ never overestimates the cost to the goal $n3$. This case may make us conjecture that if there exists a shortest path such that for each node n in the path $h(n)$ never overestimates the cost to the goal from n , then A^* finds a shortest path.

Let us consider the DAG shown in Fig. 2. Let us suppose that h is defined as follows: $h(n) = 0$ for all nodes n , namely that A^* is equivalent to the Dijkstra algorithm. The two model checking experiments do not find any counterexamples.

Let us suppose that h is defined as follows: $h(n) = 10$ for all nodes n . This h is not admissible. Moreover, there exists no shortest path such that for each node n in the path $h(n)$ never overestimates the cost to the goal from n . The two model checking experiments, however, do not find any counterexamples. Thus, the sufficient condition that A^* finds a shortest path can be relaxed.

A relaxed sufficient condition would be that there exists a shortest path such that for each node n in the path $h(n)$ plus the cost to n from the start node is less than the cost of any non-shortest path to the goal from the start. Let us suppose that $h(n2) = 5$, $h(n4) = 0$, $h(n5) = 3$, $h(n6) = 4$, $h(n7) = 0$, $h(n8) = 2$, $h(n10) = 0$ and $h(n) = 10$ for

any other node n . This h is not admissible but does satisfy the relaxed sufficient condition. The two model checking experiments do not find any counterexamples. Let us modify h a bit as follows: $h(n8) = 2$ and for any other node n $h(n)$ is the same as the last version. This modified h does not satisfy the relaxed sufficient condition. The first model checking experiment does not find any counterexamples, while the second one finds a counterexample, where A^* finds $n0 \rightarrow n4 \rightarrow n10$ that is not a shortest path. The two cases support that the relaxed sufficient condition is likely to make sense.

VI. CONCLUSION

We have reported on a case study in which A^* algorithm is formally specified in Maude and model checked with the Maude LTL model checker. Two properties have been taken into account: the termination property and the correctness property. The former says whether A^* halts, while the latter says whether A^* finds a shortest path. If the estimation h is admissible, our model checking experiments say that A^* satisfies both properties. Our model checking experiments have also suggested that the condition can be relaxed. We have then conjectured that if there exists a shortest path such that for each node n in the path $h(n)$ plus the cost to n from the start node is less than the cost of any non-shortest path to the goal from the start, A^* finds a shortest path.

Among future directions are as follows. Many path finding algorithms have been proposed. One piece of our future work is to formally specify such algorithms in Maude and model check them with the Maude LTL model checker, coming up with a framework in which such algorithms can be systematically analyzed. Another piece of our future work is to apply Maude and the Maude LTL model checker to path planning for multiple robots or vehicles, which could contribute to a near future autonomous vehicle world.

REFERENCES

- [1] P. E. Hart, et al., "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, pp. 100–107, 1968.
- [2] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.
- [3] M. Clavel, et al., *All About Maude*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4350.
- [4] A. K. Saberi, et al., "Analysis of path planning algorithms: a formal verification-based approach," in *ECAL 2013*, 2013, pp. 232–239.
- [5] N. Maragos, et al., "A formal and executable model for path finding," in *SEEFM 2003*, 2003, pp. 129–139.
- [6] A. Ulusoy, et al., "Optimality and robustness in multi-robot path planning with temporal logic constraints," *I. J. Robotics Res.*, vol. 32, pp. 889–911, 2013.

PAT approach to Architecture Behavioural Verification

Nacha Chondamrongkul*, Jing Sun[†], Ian Warren[‡]

Department of Computer Science

The University of Auckland

Auckland, New Zealand

*ncho604@aucklanduni.ac.nz

[†] jing.sun@auckland.ac.nz

[‡] i.warren@auckland.ac.nz

Abstract—Software architecture design plays a vital role in software development, as it gives an overview of how the software system should be constructed and executed at runtime. The verification of software architecture design is hence important but it is an error-prone task that heavily relies on knowledge and experience of the software architect, especially for a large software system that its behaviour is complex. Automated verification can be a solution to this problem, however, the specification language must be expressive enough to describe the behaviour of different design entities. This paper presents an enhancement of an architecture description language supported by PAT. The enhancement aims to improve the expressiveness of the language, in order to support the automated behaviour verification of software architecture design. With this enhancement, different behaviour of specific component and connector can be thoroughly checked and traced. The implementation of this enhancement is presented to demonstrate how the standard model checking engine such as PAT can be extended to support an architecture description language. We evaluated our approach with a case study and the result is presented.

Index Terms—Software Architecture, Architecture Description Language, Model Checking, Linear Temporal Logic

I. INTRODUCTION

Software architecture design gives an overview of how the software system is implemented and works. If the software architecture design is made incorrectly, it can cause the project to fail or delay due to design re-correction, therefore the verification is a significant task. However, the software architecture designs are usually represented by informal notations, such as graphical diagram and text. The design interpretation can hence be inconsistent and the verification process is an error-prone and time consuming task, even to those with extensive experience and knowledge. If the software architecture design can be formally defined, the verification task can be automated. Therefore, applying the formal methods to the software architecture design would be a useful approach to this problem.

Many architecture description languages (ADL) have been proposed to formally define the software architecture design model such as [1], [2], [3], and [4]. With the formal model in ADL, different properties can be defined and automatically

verified with the model checker. Allen and Garlan [1] proposed Wright, an ADL that allows connections in software architecture design to be formally defined in communicating sequential process (CSP). The formal format of design model allows to check the architectural compatibility among connections. However, the behavioural property definition and verification had not been completely addressed so there are number of works that aim to fulfil this. Darwin [5] was proposed to allow behavioural properties to be defined in the linear temporal logic (LTL) and use LTSA [2] as a model checker to verify them. Defining and verifying behaviours in the evolving software system is a challenge. Oquendo [3] presented π -Method with an ADL based on π -calculus. The ADL for π -Method helps to formally define evolvable software system in both structural and behavioural view. In addition, the refinement model can be defined to check and preserve the behavioural properties.

Some works have applied process algebra to formalize specific behaviour in the software system because of its expressiveness in describing system behaviour. Aldini et al. [6] presented a guideline that includes a principle of formalizing system behaviour into process algebra. The manual formalization from the design model to ADL has been an obstacle to making it widely used by the software engineers, due to the fact that the majority of them do not have background knowledge in the formal methods and the verification output from model checker can be difficult to understand. Therefore, the degree of formality needs to be balanced with the practicality. Some approaches, such as Bose et al. [7], Baresi et al. [8] and CHARMY [4], hence provide a feature that translates the input model in graphical notation into formal language that can be automatically checked. While, some approaches, such as Arcade [9], aim to make the verification output from the model checker more readable. The graphical abstraction may promote the practicality and understandability of using formal ADL, but the ambiguity might occur from the lack of complete semantic mapping between the graphical input and the model checking input. In addition, most of the existing approaches use standard model checking engines that is not designed with the architecture design concept. For example, Wright uses FDR as a model checker, while Darwin and

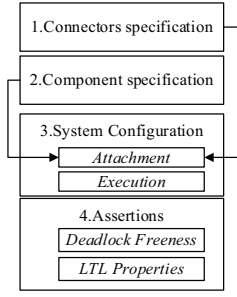


Fig. 1. Overall Approach

CHARMY uses LSTA and SPIN [10] respectively. As a result, the constructed state model is usually not optimized in term of understandability and scalability [11].

This work aims to enhance the expressiveness of Wright# [12], an architecture description language that is supported by PAT [13]. Wright# is an extension of Wright with the support of architecture styles reusing. This enhancement provides an expressive way to describe the execution of software system through system, component and connector specification. The overall approach can be found in Figure 1, which shows the specification that consists of four major parts. The connector can be specified according to the architecture styles. The component specification includes components involved in the system under design. In system configuration, the connector instances are created and attached to the component before we define how the system is executed through the process that initiates different components. The assertion is where the number of properties are defined for checking specific behaviour in the system. Deadlock freeness, a standard PAT feature, can also be used to verify the software architecture design. As the original Wright# produces a vague event labels that are difficult to trace. Therefore, the event labelling is optimized to clearly represent specific events in the connectors and components. As a result, the LTL properties can be defined to check the specific behaviours of design entities and the verification output is easier to trace back. The PAT extension module is developed as a graphical interface tool to support Wright# and its enhancement. In addition, we demonstrate the expressiveness of the design specification and property definition with a software architecture design of e-commerce software system.

The remainder of this paper is organized as follows. Section 2 explains the formalization of behaviour specification in the software architecture design. Section 3 presents the implementation of tool and how we develop a module in PAT framework to support Wright#. We demonstrate our approach with an e-commerce case study in Section 4. Section 5 concludes this paper and addresses the directions of future work.

II. FORMALIZATION OF ARCHITECTURAL DESIGN

In this section, we present the formalization of behaviour specification of software architecture design in Wright#. Wright# notation aims to define software architecture design

in component and connector view. CSP, a process algebra notation, is used to formally describe the interactive behaviours of component and connector. We extend the PAT tool to support this ADL and transform it into native CSP that forms Labelled Transition System (LTS). With LTS, the desired behaviours of software system can be automatically checked through LTL, as well as the deadlock situation.

A. Formal Modelling

Wright# is an ADL that is inspired by Wright with four basic design entities namely the component, connector, port and role. The component represents a computational unit, while the connector represents linkage between the components. The connector can includes one or more roles representing how the communication works. The component contains a number of ports that can be attached to one or more roles defined by different connectors.

There are three parts of design model to be described in ADL namely connector definition, component definition and system configuration. Each definition of connector is corresponding to different type of communication according to the architecture style. The component are defined to represent actual component and port within the software system under design. The system configuration defines how component and connector are attached, as well as the execution process. These definition contains defined processes that are based on CSP. Table I shows the syntax of process expression that can be used to describe the processes within the software architecture design.

TABLE I
PROCESS EXPRESSION SYNTAX

$e \rightarrow P$	Event prefixing
$ch!p \rightarrow P$	Channel output
$ch?p \rightarrow P$	Channel input
$P \parallel Q$	Parallel process
$P \parallel\!\!\parallel Q$	Interleaving process
$P < \star > Q$	Coupling process
<i>Stop</i>	deadlock stop
<i>Skip</i>	terminate successfully

An event represents an abstract observation of a software system. It may refer to certain system state at a given time. Event prefixing hence represent a circumstance when an event e occurs then process P is executed next. Channel output and input are used to send and receive data from its executing environment respectively. Let ch be a channel and p is data to be sent or received; P is a process to be executed next. A pair of processes can be defined as parallel, interleaving and coupling. The coupling operator does not exist in native CSP or CSP# but it is added to the syntax to represent coupling process between components. Let P and Q be a process and $P < \star > Q$. When the process P is triggered, it contains a sequence of event that an event sequentially calls the process Q to execute and return back to where it is called on process P . In order to define coupling, process P must contain an event *process*, which is when the coupling process Q is called to execute.

1) *Connector Definition*: The connectors are firstly defined to manifest how roles interact together. The processes for role are defined with the sequence of events. The channel is used to represent communication between different roles, which results in transition between events. Below is a sample code that conveys the communication for the client-server structure. The channel *req* is used to make a request from the client to the server and channel *res* is used to return response message from the server to the client.

```
connector CConnector{
  role client(j) = request → req!j → res?j
    → process → client(j);
  role server() = req?j → invoke
    → process → res!j → server(); }
```

The connector for publisher-subscriber styles can be defined as shown below, where a channel *pub* is used to broadcast data from the publisher to the subscriber.

```
connector PConnector{
  role publisher(j) = process → pub!j → Skip;
  role subscriber() = pub?j → process → subscriber(); }
```

2) *Component Definition*: The component definition contains a set of port definition. Each port has a process defined as the sequence of event that the port performs internally within the component. The script below shows two sample component namely *SPClient* and *SPServer*. The *SPClient* component has a *test* port defined and the *SPServer* has *run* port defined.

```
component SPClient {
  port test() = precheck → output → test(); }
component SPServer {
  port run() = invoke → execution → run(); }
```

3) *System Configuration*: The system configuration contains details of how components interact among each other and can be defined as follow. Firstly, the instance of connector needs to be created with the *declare* statement based on a defined connector. Secondly, the ports of connector are attached to one of more roles of connector instances using *attach* statement. If more than one roles are attached, a process expression composed of multiple role and process operator can be defined. Lastly, the *execute* statement declares how the system are executed with a process expression.

```
system SampleCS {
  declare cslink = CConnector;
  attach SPClient.test() = cslink.client();
  attach SPServer.run() = cslink.server();
  execute SPServer.run() || SPClient.test(); }
```

Careful readers may notice an event *process* defined at the role processes. This event triggers an execution of a process defined on the attached port. According to the sample system configuration shown above, the LTS is illustrated in Figure 2, which the events of port is shown in italic.

The *process* event also serves as the point of execution when the coupling process is defined. The coupling process may occur in many situation. For example, the multi-tier architecture that a tier can accept a request and consequently make a request to the upper tier. Another example is in Service-oriented architecture when a service is invoked and

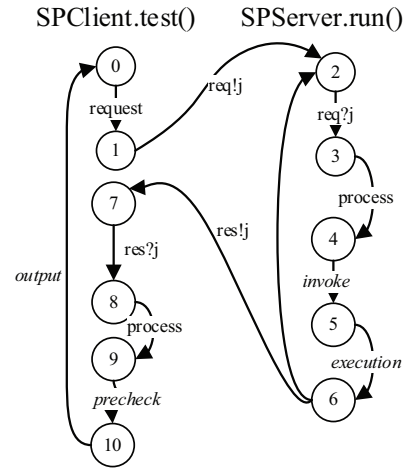


Fig. 2. LTS for the sample clien-server system

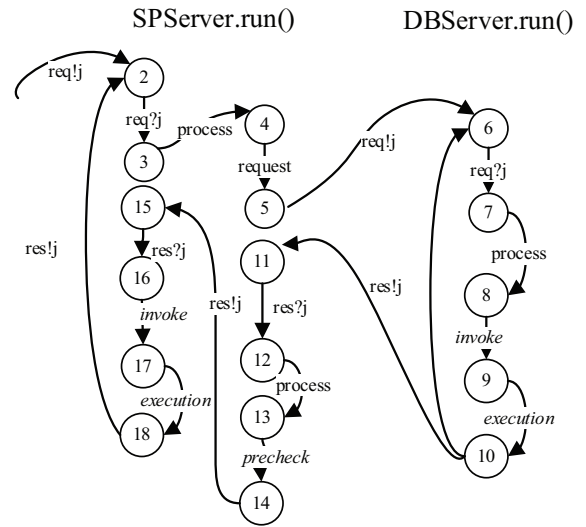


Fig. 3. LTS for the sample coupling process

calls another service. Below is a sample of attached coupling role processes. The *run* port of *SPServer* is attached to a coupling of two roles from different client-server connectors namely *cslink* and *dlnk*, which calls the component *SPServer* and *DBServer* respectively where *DBServer* is a component. In this case, the role processes are nested within one another as LTS shown in Figure 3. This coupling feature eases the complexity of defining the coupling process in native CSP.

```
attach SPServer.run() = cslink.server() < * > dlnk.client();
attach DBServer.run() = dlnk.server();
```

B. Behaviour Verification

After the software architecture design model is defined with ADL, different assertions representing the query about system behaviours can be defined. PAT supports a number of different assertion checking including linear temporal properties and deadlock freeness.

1) *Deadlock Freeness*: Deadlock is a situation when the software system can not progress further towards completion; so that the entire system halts and waits indefinitely. A well known scenario is when components wait for the mutual exclusive resource. In the software system, deadlock occurs when components call each other as circle, so the port that initializes the process loops back to itself. More concrete examples will be provided in the case study section. With the sample model explained in the previous section, a deadlock can be checked against a defined system using the *deadlockfree* statement as shown below.

assert *SampleSystem* *deadlockfree*;

2) *Linear Temporal Properties*: A full set of linear temporal logic is supported by PAT. Therefore, operators such as \Box (always), \Diamond (eventually), X (next), R (release) and U (until) can be included in the linear temporal logic defined for checking properties. Let F be a LTL formula, the assertion syntax for defining LTL properties is as follows.

assert *SampleSystem* $\models F$;

In order to support expressiveness of defining system behaviour, the property can be implicitly defined to check the behaviour of a specific component or connector.

Let *Comp* be any component, *Prt* is port of that component and *Evt* is one of the event defined in the port process. F is a LTL formula to check the behaviour of the component:

$$F = [Comp.Prt.Evt] \mid \Box F \mid \Diamond F \mid X F \mid U F \mid R F$$

Let *Comp* be any component, *Conn* be a connector, *Rle* be an attached role and *Evt* be one of the event defined in the role process. F is a LTL formula to check the behaviour of connector:

$$F = [Comp.Conn.Rle.Evt] \mid \Box F \mid \Diamond F \mid X F \mid U F \mid R F$$

For example, $\Box\Diamond SP_{Server}.run.execution$ expresses a property to check if the *execution* event always eventually occurs at the *SPServer* component. $\Diamond DB_{Server}.dblink.client.request$ expresses a property to check if the *request* event at attached client role of *DBServer* component eventually occurs.

III. TOOL IMPLEMENTATION

To support editing architecture design model in ADL and automated behaviour verification, the PAT ADL module is developed by extending PAT framework. This module includes parser that helps to parse the ADL code into objects representing different entities of software architecture design model. The parser in the original PAT tool was developed using ANTLR version 3, where different parts of parsing code in C# are merged inside the grammar file. This style of development is difficult to make any extension and maintenance. Therefore, we adopt ANTLR version 4.0, where the source code of language parsing can be separated from the grammar file. The complete source code of PAT ADL can be found at <https://bit.ly/2Vc855I>.

The overall process performed by ADL module can be illustrated in Figure 4. The editor tool allows users to edit ADL file according to the syntax explained in the previous

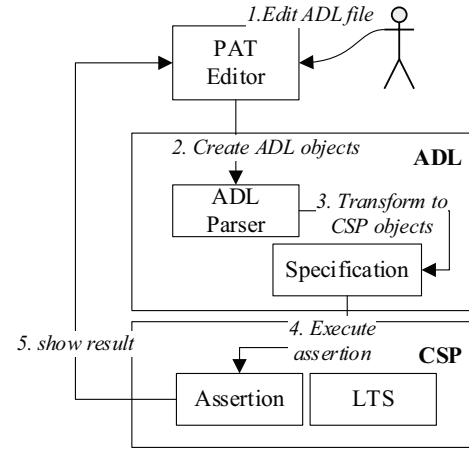


Fig. 4. Overall Process of the ADL module

section. When the user makes a verification, the source code in ADL is processed by the PAT ADL in the background and return the result back to the user interface on the editor tool. We developed a grammar file based on the CSP# and used ANTLR to automatically generate a parser program. The parser program helps to read a source code in ADL and the visitor program is developed to convert different ADL statements into objects representing entities such as component, connector, system configuration, port, role, attachment and assertion. With *Specification* module, ADL objects are later automatically transformed into PAT native objects representing the CSP processes. The ADL to CSP transformation can be briefly explained as follows.

- One process is defined corresponding to each attached role of a defined port.
- One process is defined to represent a defined port, which calls the attached role process according to the expression defined in the *attach* statement.
- One process is created for a defined system and call port processes according to expression defined in the *execute* statement.

For example, the sample client-server model explained in the previous section can be transformed into native CSP as shown below. Two channels namely *cslink_req* and *cslink_res* are defined for requesting and responding message. *SPClient_cslink_client* process is defined for the client role of *cslink*, and *SPClient_test* process is defined for the *test* port. Two processes namely *SPServer_cslink_server* and *SPServer_run* are defined in the same way for server. The *SampleSystem* process is defined to represent the main system process. The *LTS* sub-module helps to model LTS according to CSP and encapsulate the transition model. The transition model allows *Assertion* sub-module to traverse according to the depth-first search and breadth-first search algorithm, in order to make a verification. The verification result is displayed on the verification window of the editor tool.

```

channel cslink_res 1;
channel cslink_req 1;
SPClient_cslink_client(j) = (
  SPClient_cslink_client_request
  → cslink_req!j → cslink_res?j
  → (SPClient_cslink_client_result
  → (SPClient_cslink_client_process
  → (SPClient_test_precheck
  → (SPClient_test_output → SPClient_cslink_client(j)))));
SPClient_test() = SPClient_cslink_client();
SPServer_cslink_server() = cslink_req?j
  → (SPServer_cslink_server_invoke
  → (SPServer_cslink_server_process
  → (SPServer_run_invoke
  → (SPServer_run_execution
  → (SPServer_cslink_server_return
  → cslink_res!j → SPServer_cslink_server()))));
SPServer_run() = SPServer_cslink_server();
SampleSystem() = (SPServer_run() || SPClient_test());

```

IV. CASE STUDY

We select a part of real-world e-commerce software system to demonstrate and evaluate the practicality of our approach. The software architecture design of this system is shown (as UML component diagram) in Figure 5. The software system allows user to browse catalogue of the products, order and make a purchase on-line through the web store or mobile store. When the users make a purchase, the order manager component keeps the record of order and fetch the product from the inventory through the inventory control component. The inventory control automatically locates the ordering product from the warehouse and send details to the shipping control component. The shipping control allows packaging officer to prepare shipping package and log the shipping package for courier.

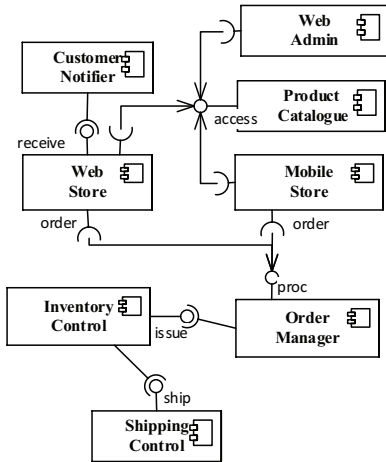


Fig. 5. Component diagram for E-commerce system

In this case study, we use the client-server and publisher-subscriber connectors as defined in the previous section. The model in ADL is partly shown below. The complete model can be found at <https://bit.ly/2EWJGCg>. The ports are defined according to the component diagram shown in Figure 5 along with the port processes that simulate what the components

perform. Some port may omit the process details such as *ProductCatalogue* and *OrderManager* component. The system configuration of this system declares 4 client-server connector and 1 publisher-subscriber connector. The number of attachments are defined to link components together. The numbers passing in as a parameter for role processes represent the signature of data passing between the component and connector. The *proc* port of *OrderManager* is attached to two roles as a coupling process: *purchasing.server()* and *issuing.client()*. This is because the order can only be processed successfully when the inventory finished fetching the product, otherwise the order is rejected. The *purchasing.server()* represents the internal process for processing order within the *OrderManager* component, while the *issuing.client()* requests to the service on *InventoryControl* component to fetch the product. The *issue* port of *InventoryControl* component is attached to two roles as a parallel process. The *issuing.server()* represents the internal execution of *InventoryControl* component, while *shipping.client()* represents a request to manage the shipping details of the *ShippingControl* component. The *execute* statement defines all port processes to be executed in parallel. The execution can be modified to focus on checking some particular scenarios in response to the functionality of the software system, such as when the product is ordered and when the product catalogue is updated.

```

component WebStore {
  port browse() = render → output → browse();
  port order() = commit → email → order();
  port receive() = acknowledge → display → receive(); }
component MobileStore {
  port browse() = render → output → browse();
  port order() = commit → email → order(); }
component WebAdmin {
  port manage() = result → manage(); }
component CustomerNotifier {
  port alert() = promo → send → alert(); }
component InventoryControl {
  port issue() = locate → fetch → issue(); }
component ShippingControl {
  port ship() = inform → log → ship(); }
...
system Shopping {
  declare purchasing, issuing = CSConnector;
  declare shipping, cataccessing = CSConnector;
  declare newswire = PSCConnector;
  attach WebStore.order() = purchasing.client(99);
  attach MobileStore.order() = purchasing.client(98);
  attach WebStore.receive() = newswire.subscriber();
  attach CustomerNotifier.alert() = newswire.publisher(77);
  attach WebStore.browse() = cataccessing.client(99);
  attach WebAdmin.manage() = cataccessing.client(98);
  attach ProductCatalogue.access() = cataccessing.server();
  attach OrderManager.proc() = purchasing.server()
    < * > issuing.client();
  attach InventoryControl.issue() = issuing.server()
    || shipping.client(88);
  attach ShippingControl.ship() = shipping.server();
  execute WebStore.order() || MobileStore.order()
    || WebStore.browse() || WebAdmin.manage()
    || ProductCatalogue.access() || OrderManager.proc()
    || InventoryControl.issue() || ShippingControl.ship();

```


Three assertions are defined as shown below. The first assertion helps to check if the software system design can leads to a deadlock. If the deadlock is found, the verification shows an invalid result with a counterexample, which gives a sequence of events leading how deadlock can occur.

```
assert Shopping deadlockfree;
assert Shopping = ◇WebStore.purchasing.client.process;
assert Shopping = □(OrderManager.purchasing.server.process
→ ◇WebStore.order.email);
```

With the system configuration above, the deadlock does not occur so it outputs a valid result. However, we can demonstrate when deadlock occurs by changing the attachment of the *ShippingControl.ship()* port to the *shipping.server()* < * > *issuing.client()*. This makes the components call each other in a loop. The verification result can be seen in Figure 6. The event labels identify both component, connector, role and port that are involved in the deadlock. The second assertion makes use of the behaviour checking on the connector. It checks if the *process* event of *client* role is eventually triggered at the attached *purchasing* connector on the *WebStore* component. The third assertion combines the behaviour checking on both the component and connector, as it checks if every time the order is processed, the email will always be sent out to the customer. The results of these two LTL properties are valid. The verification statistic of these three assertions including number of states, number of transitions, total time usage and estimated memory usage can be found in Table II. As can be seen from the table, the number of visited states, memory usage and total time are relatively low.

```
*****Verification Result*****
The Assertion (Shopping() deadlockfree) is NOT valid.
The following trace leads to a deadlock situation.
<init -> InventoryControl_shipping_client_request -> shipping_req188 -> shipping_req788 ->
ShippingControl_shipping_server_invoke -> ShippingControl_issuing_client_request -> issuing_req188 ->
issuing_req788 -> InventoryControl_issuing_server_invoke -> InventoryControl_issuing_server_process ->
WebStore_purchasing_client_request -> purchasing_req199 -> purchasing_req799 ->
OrderManager_purchasing_server_invoke -> OrderManager_issuing_client_request -> issuing_req199
```

Fig. 6. Deadlock Result from PAT

TABLE II
VERIFICATION STATISTIC

Assertion	State#	Transition#	Time (sec)	Memory
Deadlock	25	24	0.0059828	8664 KB
LTL 1	16	24	0.0105143	8696 KB
LTL 2	159	278	0.0137552	42280 KB

V. CONCLUSION

We present an enhancement to Wright# ADL that supports the formal behaviour modelling in software architecture design. The language allows users to expressively define and verify the behaviour of components and connectors. The implementation of a PAT extension module to support Wright# and our enhancement is presented, in order to demonstrate how the standard model checker can be extended to support an ADL. We evaluate our approach with an e-commerce

software system. Our approach can be used to clearly define the behaviour of different components and connectors in the design, as well as the interaction among them. The properties can be defined in LTL assertions to represent the desired system behaviour in response to the system functionalities. The deadlock analysis, a standard feature in PAT can be used. The event labels in a counterexample is informative enough to identify involved design entities that cause invalid behaviour. We found that the state space are relatively low but more evaluation need to be taken to prove the scalability.

For the future work, we plan to integrate this approach with other techniques such as ontology reasoning [14], in order to fulfil the semantics of the architecture design in the verification process. As the ontology representation is rich of semantic constrains that can help to verify and maintain the structure consistency in the design model before its behaviour is checked. More case studies in the real world could be used to evaluate the practicality and scalability of our approach. As the behaviours can be formally defined, it could be interesting to use it to detect the design smells or anti-pattern based on the system behaviours.

REFERENCES

- [1] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 3, pp. 213–249, Jul. 1997.
- [2] J. Magee and J. Kramer, *Concurrency: State Models & Java Programs*. New York, NY, USA: John Wiley & Sons, Inc., 1999.
- [3] F. Oquendo, "π-method: A model-driven formal method for architecture-centric software engineering," *ACM Sigsoft Software Engineering Notes*, vol. 31, pp. 1–13, 05 2006.
- [4] P. Pelliccione, P. Inverardi, and H. Muccini, "Charmy: A framework for designing and verifying architectural specifications," *IEEE Transactions on Software Engineering*, vol. 35, pp. 325–346, 2009.
- [5] J. Magree, "Behavioral analysis of software architectures using Itsa," in *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*, May 1999, pp. 634–637.
- [6] A. Aldini, M. Bernardo, and F. Corradini, *A Process Algebraic Approach to Software Architecture Design*. Springer Publishing Company, Incorporated, 2014.
- [7] P. Bose, "Automated translation of uml models of architectures for verification and simulation using spin," in *Proceedings of the 14th IEEE International Conference on Automated Software Engineering*, ser. ASE '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 102–.
- [8] L. Baresi, C. Ghezzi, and L. Zanolin, *Modeling and Validation of Publish/Subscribe Architectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 273–291.
- [9] K. S. Barber, T. Graser, and J. Holt, "Providing early feedback in the development cycle through automated application of model checking to software architectures," in *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, Nov 2001, pp. 341–345.
- [10] G. Holzmann, *Spin Model Checker, the: Primer and Reference Manual*, 1st ed. Addison-Wesley Professional, 2003.
- [11] P. Zhang, H. Muccini, and B. Li, "A classification and comparison of model checking software architecture techniques," *Journal of Systems and Software*, vol. 83, no. 5, pp. 723 – 744, 2010.
- [12] J. Zhang, Y. Liu, J. Sun, J. S. Dong, and J. Sun, "Model checking software architecture design," in *2012 IEEE 14th International Symposium on High-Assurance Systems Engineering*, Oct 2012, pp. 193–200.
- [13] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "Pat: Towards flexible verification under fairness," ser. Lecture Notes in Computer Science, vol. 5643. Springer, 2009, pp. 709–714.
- [14] N. Chondamrongkul, J. Sun, and I. Warren, "Ontology-based software architectural pattern recognition and reasoning," in *30th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, June 2018, pp. 25–34.

Leveraging Rigorous Software Specification Towards Systematic Detection of SDN Control Conflicts

Xin Sun and Lan Lin
Ball State University, Muncie, Indiana 47306, USA
{xsun6, llin4}@bsu.edu

Abstract—This paper leverages a well-established, rigorous method for software specification to approach a unique problem introduced by the emerging software-defined networking (SDN) paradigm, i.e., the potential control conflict arising from running multiple SDN apps in the same network. As individual SDN apps have different optimization objectives and each assumes full control of the network, their interaction is often unpredictable and can destabilize the network as a result. We propose a theoretical modeling framework for systematically detecting such conflicts, which is deeply rooted in automaton theory and software engineering. The key novelty and strength of our approach is its ability to model and reason about the interaction of multiple SDN apps precisely (with the capability of identifying when and how conflicts may occur), proactively (prior to running the apps), and without the knowledge of the apps' implementation details. To the extent of our knowledge our work is the first to adapt rigorous software specification to the constructive, formal modeling of SDN apps running on a network topology, and through a formal treatment not only straightforwardly detects and locates such conflicts but also examines and analyzes important network properties (e.g., safe operational regions) of interest to network managers.

I. INTRODUCTION

This paper reports an application of rigorous software specification to the emerging software-defined networking (SDN) paradigm, to approach a unique problem introduced by “SDN apps” (also called network functions), which are software applications running on top of the SDN controller platform, offering a variety of functionalities such as load-balancing, power-saving, quality of service, access control, WAN optimization, network virtualization, to name a few (a comprehensive survey is presented in [1]).

Because (i) the apps are created by different developers (virtually anyone can develop and release them; marketplace exists today for selling and buying SDN apps, e.g., Hewlett-Packard App Store [2]), (ii) each app typically manages/optimizes a single aspect of the network (e.g., performance, security, resiliency, energy usage, etc.), and has a single optimization objective, and (iii) each app assumes full control of the whole network, they may seek to change the underlying network in conflicting ways. The interaction of their conflicting outputs can be unpredictable and, as a result, destabilize the network (a case study of such conflict is presented in Sec. III).

In our preliminary work [3] a fine-grained approach was proposed that models the SDN apps and their interactions using deterministic finite state automata. However, derivation of the automata was completely *manual*, and the conflicts were manually identified afterwards based on human insight. The modeling process was tedious with much trial and error, and we were not able to prove the correctness of either the derived automata or the located conflicts.

This paper presents an advanced approach that significantly extends our previous work, with the following contributions:

- 1) We adapted a well-established, rigorous method for software specification (i.e., *sequence-based specification* [4, 5, 6, 7]) to systematically derive a formal (automaton) model for each SDN app that runs on a network topology (Sec. IV). The adapted method is inherently rigorous, systematic, and constructive, and does not require knowledge of the implementation details of the apps; as such, we believe the method is very practical.
- 2) We developed a theoretical framework for analyzing the interaction of multiple SDN apps running in parallel (Sec. V). With our new theory, important network properties of interest to network managers, such as the safe operational region (i.e., network states under which multiple SDN apps can run free of conflict), can for the first time be formally defined and precisely analyzed. We demonstrated how potential control conflicts can be straightforwardly and systematically detected (Sec. VI).

II. RELATED WORK

The problem of control conflicts caused by running multiple independently-developed SDN apps, and the resulting destabilization of the network, has recently started to receive attention from the research community. Corybantic [8] and Athens [9] take a coarse-grained approach that resolves potential conflicts at run-time (i.e., when the network is in operation). They let individual SDN apps generate “proposals” for network configuration changes, and then require each app to evaluate all proposals, based on some predefined policy or voting mechanism. While this approach can successfully resolve conflicts, it does so with significant costs: (i) it requires all SDN apps to implement the additional functionality of generating and evaluating proposals; (ii) it only selects a single proposal at a time, which can leave out potentially better solutions that combine multiple conflict-free proposals, leading to sub-optimal network configurations; (iii) it cannot identify the root cause of the conflicts. Bairley and Xie [10] take a similar approach (it thus suffers from similar drawbacks), except that it seeks to combine multiple proposals to form a globally optimal configuration using an evolutionary approach. In contrast, our approach does not have any of those drawbacks.

Prior works [11, 12] concern individual-flow-level forwarding behavior and can detect policy violations (such as black holes and loops) at that level; however, none of them can detect the network state oscillation caused by conflicts of running multiple SDN apps, since state oscillation is not a violation of flow policy (it is a higher-level issue).

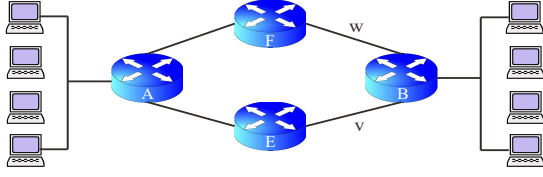


Fig. 1. The Network Topology Used in the Case Study

III. CONTROL CONFLICTS AND NETWORK STATE OSCILLATION: A CASE STUDY

A software-defined network typically has a three-tier architecture [13]. The bottom layer, also called the data plane, consists of “dumb” switches and other hardware boxes that primarily focus on packet streaming. The middle layer, also called the controller or the network operating system, is a software platform that directly manages the hardware boxes and offers an abstraction of the network resources via a set of application programming interfaces (APIs). The APIs in turn enable the development of network functions, also called “SDN apps”, which form the top layer.

An SDN app generally seeks to optimize some aspect of a network, by modifying the network state. The state of a network includes multiple variables, such as traffic load, routing paths, power state of devices, up/down state of links, etc. [14] SDN apps control the network state by issuing commands to the controller via the northbound API; the controller then compiles each command down to a set of new configurations to be installed on the data plane.

We observed that, when multiple SDN apps are trying to control the network state, conflicts may occur. For the purpose of demonstrating both the problem and our solution in a tangible and unambiguous manner, we now present a concrete case study involving two popular types of SDN apps that have been extensively researched (e.g., [15, 16, 17]): a *power-saving* app and a *load-balancing* app, running on a toy network as depicted by Fig. 1. End hosts attach to Routers A and B. Between them there are two hosts routing paths: one goes through Router E and the other F. The traffic load on each link (w or v) is modeled as a step function that has three states: high (H), low (L), and zero (Z). Assume all devices and links have the same capacity. The two apps work in the following way:

Power-saving app: it seeks to aggregate traffic to one of the two paths (without loss of generality, assuming it’s always the top path), and then turn off Router E to reduce energy use. To realize this objective, it checks the utilization rates of Links w and v every M seconds, and whenever the utilization rates of both links are low or zero for two consecutive cycles, it issues a command to the controller to turn off E and route all future packets to the top path.

Load-balancing app: it seeks to spread traffic across all possible paths, to minimize the load of any link. It checks the utilization rates of w and v every M seconds. Whenever one is high and the other is low or zero for three consecutive cycles, it issues a command to the controller to route a larger fraction of new flows entering the network in the next M seconds through the less congested path. It stops issuing such commands once the utilization rates of both links become high, or both become low or zero (i.e., the load is balanced).

Intuitively, when the two apps run on the same network, control conflict might occur, as one seeks to aggregate traffic to a subset of paths, while the other seeks to spread traffic evenly on all paths. However, the simple intuition is unable to tell precisely when and how control conflict may arise, and the consequences of the conflicts. The problem can get much more complicated, as the conflict may be caused by more than two apps, and the network may oscillate among more than two states.

IV. MODELING INDIVIDUAL SDN APPS

This section first presents a formal model for individual SDN apps, and then describes our approach for deriving the model. The two SDN apps described in Sec. III are used as running examples to illustrate the modeling process.

A. An Automaton-Based Model

Our model of an SDN app running on a network topology follows the conventional definition of a Moore machine.

Definition 1 (Moore Machine): A *Moore machine* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, \nu, q_0)$, where Q is a finite set of states, Σ is an input alphabet, Γ is an output alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, $\nu : Q \rightarrow \Gamma$ is the output function, and $q_0 \in Q$ is the starting state.

Specifically for the SDN apps, they can be modeled as a subset of Moore machines, of which the output alphabet includes the two special responses: 0 (the null response) and ω (the illegal response). Additionally, the model only needs to include states that are reachable from the initial state; non-reachable states can be safely ignored as they will never be realized. To model reachable states, we need to extend the transition function of a Moore machine to input sequences, as follows: $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ is defined by (1) $\hat{\delta}(q, \lambda) = q$, where λ is the empty input sequence; and (2) $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ for all $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.

Definition 2 (Network Function Moore Machine): A *network function Moore machine* is a Moore machine $(Q, \Sigma, \Gamma, \delta, \nu, q_0)$ satisfying (1) $\{0, \omega\} \subset \Gamma$, and (2) for any $q \in Q$ there exists $w \in \Sigma^*$ such that $\hat{\delta}(q_0, w) = q$.

Here Q represents the set of *software states* of the SDN app (note that these are different from the *network states*); Σ represents inputs to the SDN app, which includes the network states such as link load and topology; Γ represents outputs from the SDN app, which are commands to the SDN controller that seek to change the network state; 0 and ω are special outputs representing the null output, and the illegal output (for an input sequence not possible to occur – this is defined for completeness purposes). The second condition ensures that any state in Q must be reachable from the initial state q_0 .

B. A Rigorous Approach for Deriving the Automaton

Our approach for deriving the Moore machine of a given SDN app is rigorous (based on the automaton theory), systematic (we offer a systematic process to follow), and constructive (the state machine will be discovered at the end of the process). It is based on a well-established rigorous method for software specification, i.e., *sequence-based specification* [7, 6, 5, 4]. The input to our approach is an informal description of how the SDN app is supposed to work on the given network topology

(this description is termed “functional requirements” in the field of software engineering). The output is a Moore machine representing the working mechanism of the SDN app. The approach has three key steps: identification of stimuli and responses, sequence enumeration, and automaton construction.

1) Step 1: Identification of stimuli and responses:

We first identify a list of *stimuli* (i.e., inputs) and *responses* (i.e., outputs) of the SDN app running on the given network topology. The power-saving app and the load-balancing app share a common set of inputs: $\{H_w H_v, H_w L_v, H_w Z_v, L_w H_v, L_w L_v, L_w Z_v, Z_w H_v, Z_w L_v, Z_w Z_v\}$, where H , L , and Z indicate link utilization is high, low, or zero, respectively, and the subscript indicates which link (i.e., w or v). Each input contains information about the utilization rates of both links. Outputs of the power-saving app are $\{OnE, OffE, 0\}$, where $OnE/OffE$ represents turning on/off the switch E (a request to turn on/off E , rather than a successful command or operation, as such request may be overridden [3]), respectively, and 0 represents the *null* response, i.e., no output issued by the SDN app that potentially changes the network’s state. Outputs for the load-balancing app are $\{FtoE, EtoF, 0\}$, where $FtoE/EtoF$ represents moving flows from F/E to E/F , respectively.

2) Step 2: Sequence enumeration: We start with an informal description of the requirements for each SDN app running on the given topology, and tag (number) them to facilitate tracing decisions we have made (in the specification process) to the tagged requirements or derived requirements (as a by-product the process also leads to the discovery of derived requirements that were not originally stated, and the resolution/correction of inconsistent/incorrect requirements). Requirements for the power-saving app and for the load-balancing app are listed in Tables I and II, respectively. Original requirements were retrieved from descriptions of the functions in [3]. Tags that begin with D indicate new requirements we derived in the specification process (following the same assumptions as implied by [3]).

Next we perform the key step of sequence-based specification, called *sequence enumeration*, to discover/construct every detail of the state machine. We enumerate all finite sequences of stimuli (inputs) in length-lexicographical order (i.e., first by length, and within the same length lexicographically), and for each enumerated sequence make two decisions:

- Response mapping. We map the sequence of inputs to a response (output) of the network function. The response is the output the network function produces in response to the very last input in the sequence, given the input history. For instance, the sequence $L_w L_v . L_w L_v$ (we concatenate inputs with dots) is mapped to the response $OffE$ by the power-saving function by Requirement 2 (Table I). We introduce two special responses in theory: the *null* response, denoted by 0 , for the lack of an externally observable output (there might have been an internal state update), and the *illegal* response, denoted by ω , for an operationally unrealizable sequence of inputs (the sequence cannot occur in practice). A sequence is *illegal* if it is mapped to ω ; otherwise, it is *legal*.
- Equivalence declaration. We determine if the sequence is *Moore equivalent* to (and hence can be reduced to) a

TABLE I
REQUIREMENTS FOR THE POWER-SAVING FUNCTION

Tag	Requirement
1	Assume each link exhibits a utilization rate that can vary over time. This rate can be sampled on each cycle and is either high (H), low (L), or zero (Z).
2	Suppose a power-saving machine tries to power down E when the link utilization of both links v and w is low or zero for two consecutive cycles. Any new flows are then routed to F .
3	The machine waits for two consecutive cycles when link w is experiencing heavy load before restoring power to E .
4	The machine is designed such that the attempt to turn E off may have failed or been overridden. Thus it is treated more as a request than a command.
D1	Assume there is an attempt to turn on E at system start/initialization.
D2	Assume that turning on E is also treated as a request than a command.
D3	The latest two cycles’ link utilization for both v and w is necessary to determine if a command needs to be issued to turn on/off E .
D4	Except for at system initialization, an attempt to restore power to E has to follow a recent power off attempt for which no other restore attempt has been made, and only a zero link rate of v and two consecutive high link rates of w have been observed since that power off attempt.
D5	After a recent power off attempt that appears unsuccessful (by a non-zero link utilization of v), another two consecutive cycles of both links v and w being low or zero need to be observed to power off E again.
D6	After a recent power off attempt, if v has been observed of zero utilization it suggests the latest power off attempt might have been successful, but is subject to future observations.

TABLE II
REQUIREMENTS FOR THE LOAD-BALANCING FUNCTION

Tag	Requirement
D1	Assume load is balanced at system start/initialization.
D2	Continued balanced load does not change the (load) balanced state the function is in.
D3	The load-balancing function needs to observe three consecutive cycles of the same imbalanced load patterns before functioning.
D4	If load is observed re-balanced from an imbalanced state, the load-balancing function returns to the (load) balanced state.
D5	If heavy load switches between the two links v and w , the load-balancing function transitions to the corresponding imbalanced state (based on which link has heavy load).
D6	After observing three consecutive cycles of the same imbalanced load patterns (i.e., high w and low/zero v , or high v and low/zero w), the load-balancing function directs load from the heavy link to the light link.
D7	Load-balancing operations are issued based on the most recent three consecutive cycles’ loads only, irrespective of whether the same operation has been recently issued.

previously enumerated sequence. Two sequences u and v are *Moore equivalent* if and only if for any input sequence w , uw and vw always map to the same response by the network function. This implies u and v are mapped to the same response as well (as w could be the empty sequence). For instance, $L_w L_v . L_w Z_v$ can be reduced to the prior sequence $L_w L_v . L_w L_v$ by Requirement 2 for the power-saving function. Two equivalent sequences arrive at the same state of the underlying Moore automaton starting from the initial state. We chose to model it using a Moore machine to be consistent with the transducer model in [3], whereas in software specification *Mealy equivalence* and a Mealy machine are used as they lead

to a shorter enumeration table. One could easily transform between Moore and Mealy machines. When reducing a sequence to a prior sequence, we follow the reduction chain and get to the sequence that is itself unreduced. For instance, $H_w Z_v$ is reduced to $H_w H_v$ and not $H_w L_v$ as $H_w L_v$ is further reduced to $H_w H_v$ (for the power-saving function). A sequence is *reduced* if it is Moore equivalent to a prior sequence in length-lexicographical order; otherwise, it is *unreduced*.

One starts with the empty sequence λ . To get all the sequences of Length $n + 1$ (integer $n \geq 0$) one extends all the sequences of Length n by every stimulus, and considers the extensions in lexicographical order. This inherently combinatorial process can be controlled by two observations:

- If Sequence u is reduced to a prior sequence v , there is no need to extend u , as the behaviors of the extensions are defined by the same extensions of v .
- If Sequence u is illegal, there is no need to extend u , as all of the extensions must also be illegal (i.e., physically unrealizable).

Therefore, only legal and unreduced (also called *extensible*) sequences of Length n get extended by every stimulus for consideration at Length $n + 1$. The process continues until all the sequences of a certain length are either illegal or reduced to prior sequences. The enumeration becomes *complete*. This terminating enumeration length is discovered in enumeration, and varies from application to application.

Excerpt of an enumeration for the power-saving function is shown in Table III. We show the enumeration until Length 3 due to lack of space (the enumeration terminates at Length 4). Columns of the table are for enumerated sequences, their mapped responses, possible reductions to prior sequences under Moore equivalence, and traces to requirements. We similarly performed the enumeration for the load-balancing function, but omitted it here due to lack of space.

3) **Step 3: Construction of the automaton:** We observe that the completed enumeration from the previous step encodes a Moore machine as follows. First we retrieve all the unreduced sequences; each represents a state, whose associated output is the mapped response of the unreduced sequence (that can be read off from the table). Table IV shows the mapping from unreduced sequences in the power-saving enumeration to Moore states of the power-saving automaton. The mapping table for the load-balancing function is omitted here due to lack of space.

With this observation we are now ready to construct the automaton. For doing so, we simply map each row in the enumeration table (except the empty sequence) to a transition in the Moore machine as follows: if the prefix sequence u concatenated with the current stimulus a is reduced to the sequence w (here we treat any unreduced sequence as being reduced to itself; an equivalence relation must be reflexive), then a transition triggered by the input a goes from the state represented by u to the state represented by w . For instance, $H_w H_v . H_w H_v$ being reduced to $H_w H_v$ in the power-saving enumeration implies a transition from State r_0 to State r_0 on Input $H_w H_v$ for the power-saving automaton.

The state machines for both apps constructed in this step are shown in Fig. 2 and Fig. 3. Their equivalent formal definitions

TABLE III
EXCERPT OF AN ENUMERATION FOR THE POWER-SAVING FUNCTION
UNTIL LENGTH 3

Sequence	Response	Equivalence	Trace
λ	OnE		D1, D2
$H_w H_v$	0		D3
$H_w L_v$	0	$H_w H_v$	D3
$H_w Z_v$	0	$H_w H_v$	D3
$L_w H_v$	0	$H_w H_v$	D3
$L_w L_v$	0		D3
$L_w Z_v$	0	$L_w L_v$	D3
$Z_w H_v$	0	$H_w H_v$	D3
$Z_w L_v$	0	$L_w L_v$	D3
$Z_w Z_v$	0	$L_w L_v$	D3
$H_w H_v . H_w H_v$	0	$H_w H_v$	D3
$H_w H_v . H_w L_v$	0	$H_w H_v$	D3
$H_w H_v . H_w Z_v$	0	$H_w H_v$	D3
$H_w H_v . L_w H_v$	0	$H_w H_v$	D3
$H_w H_v . L_w L_v$	0	$L_w L_v$	D3
$H_w H_v . L_w Z_v$	0	$L_w L_v$	D3
$H_w H_v . Z_w H_v$	0	$H_w H_v$	D3
$H_w H_v . Z_w L_v$	0	$L_w L_v$	D3
$H_w H_v . Z_w Z_v$	0	$L_w L_v$	D3
$L_w L_v . H_w H_v$	0	$H_w H_v$	D3
$L_w L_v . H_w L_v$	0	$H_w H_v$	D3
$L_w L_v . H_w Z_v$	0	$H_w H_v$	D3
$L_w L_v . L_w H_v$	0	$H_w H_v$	D3
$L_w L_v . L_w L_v$	OffE		2
$L_w L_v . L_w Z_v$	OffE	$L_w L_v . L_w L_v$	2
$L_w L_v . Z_w H_v$	0	$H_w H_v$	D3
$L_w L_v . Z_w L_v$	OffE	$L_w L_v . L_w L_v$	2
$L_w L_v . Z_w Z_v$	OffE	$L_w L_v . L_w L_v$	2
$L_w L_v . L_w L_v . H_w H_v$	0	$H_w H_v$	4
$L_w L_v . L_w L_v . H_w L_v$	0	$H_w H_v$	4
$L_w L_v . L_w L_v . H_w Z_v$	0		3, D6
$L_w L_v . L_w L_v . L_w H_v$	0	$H_w H_v$	4
$L_w L_v . L_w L_v . L_w L_v$	0	$L_w L_v$	4, D5
$L_w L_v . L_w L_v . L_w Z_v$	0		D6
$L_w L_v . L_w L_v . Z_w H_v$	0	$H_w H_v$	4
$L_w L_v . L_w L_v . Z_w L_v$	0	$L_w L_v$	4, D5
$L_w L_v . L_w L_v . Z_w Z_v$	0	$L_w L_v . L_w L_v . L_w Z_v$	D6

TABLE IV
THE MAPPING FROM UNREDUCED SEQUENCES TO MOORE STATES FOR
THE POWER-SAVING FUNCTION

Unreduced Sequence	State	Output
λ	r_5	OnE
$H_w H_v$	r_0	0
$L_w L_v$	r_1	0
$L_w L_v . L_w L_v$	r_2	OffE
$L_w L_v . L_w L_v . H_w Z_v$	r_4	0
$L_w L_v . L_w L_v . L_w Z_v$	r_3	0

are omitted for lack of space.

V. MODELING JOINT EFFECT OF MULTIPLE SDN APPS

One benefit of using automata to model SDN apps is that, when two SDN apps run in parallel on the same network topology, their behavior can be modeled by the standard automaton product, which can be straightforwardly computed by applying the following definition:¹

Definition 3 (Product of Network Function Moore Machines): Given two network function Moore machines $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, \nu_1, q_{1,0})$ and $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, \nu_2, q_{2,0})$, the *product* of M_1 and M_2 , denoted by $M_1 \times M_2$, is defined

¹This operation can be easily extended to more than two SDN apps and performed in a successive way.

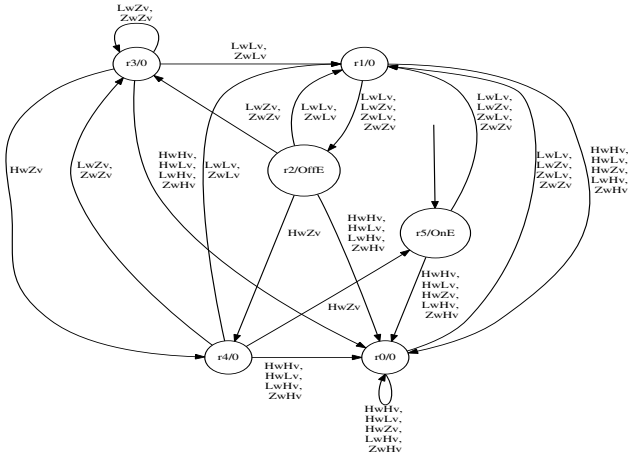


Fig. 2. The Power-Saving Moore Machine

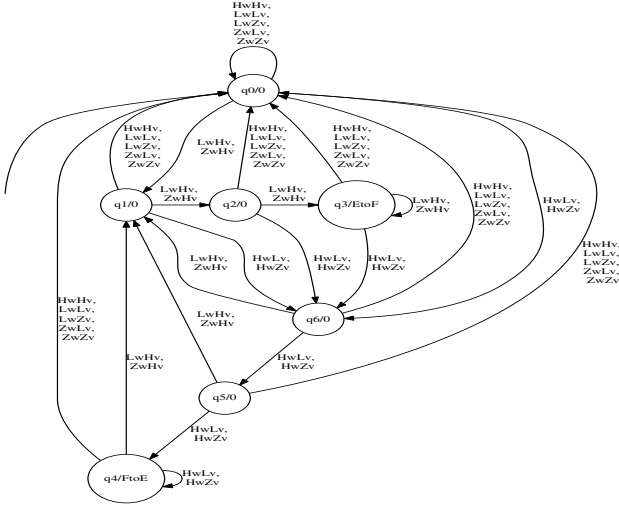


Fig. 3. The Load-Balancing Moore Machine

by $M_1 \times M_2 = (Q_1 \times Q_2, \Sigma_1 \times \Sigma_2, \Gamma_1 \times \Gamma_2, \delta, \nu, (q_{1,0}, q_{2,0}))$, where δ and ν are defined by:

$$\begin{aligned} \delta((p, q), (a, b)) &= (\delta_1(p, a), \delta_2(q, b)), \\ \nu((p, q)) &= (\nu_1(p), \nu_2(q)). \end{aligned}$$

We extend the *null* response and the *illegal* response to the product of two network function Moore machines by assuming $0 = (0, 0)$ and $\omega = (r_1, r_2)$ where either $r_1 = \omega$ or $r_2 = \omega$. An input $(a, b) \in \Sigma_1 \times \Sigma_2$ is *legal* if it is physically realizable; otherwise, it is *illegal*.

Note that we also extend the concept of being legal/illegal to inputs of a product automaton.

Example 4: The product of the power-saving Moore machine M_1 and the load-balancing Moore machine M_2 is defined by $M = M_1 \times M_2 = (Q_1 \times Q_2, \Sigma \times \Sigma, \Gamma_1 \times \Gamma_2, \delta, \nu, (r_5, q_0))$, where δ and ν are defined by:

$$\begin{aligned} \delta((p, q), (a, b)) &= (\delta_1(p, a), \delta_2(q, b)), \\ \nu((p, q)) &= (\nu_1(p), \nu_2(q)). \end{aligned}$$

Observe that $\{(a, a) : a \in \Sigma\}$ is the set of all legal inputs of M .

One interesting observation on the constructed product Moore machine is that not all its states may be reachable from the initial state. Intuitively, for a state (p_j, q_j) in the product Moore machine $M = M_1 \times M_2$ to be reachable from another

state (p_i, q_i) , there must exist paths of the same length from p_i to p_j in M_1 and from q_i to q_j in M_2 .

Definition 5 (Reachable States): Given the product $M = (Q, \Sigma, \Gamma, \delta, \nu, q_0)$ of two network function Moore machines and a set of legal inputs $I \subseteq \Sigma$, the set RS of *reachable states* of M is defined by $RS = \{q : q \in Q, \exists w \in I^*. \hat{\delta}(q_0, w) = q\}$.

This definition enabled us to develop a simple algorithm to automatically identify all reachable states in a product Moore machine, which we omitted here due to space constraint.

Definition 6 (Joint Network Function Moore Machine): Given the product $M = M_1 \times M_2 = (Q, \Sigma, \Gamma, \delta, \nu, q_0)$ of two network function Moore machines M_1 and M_2 , and a set of legal inputs $I \subseteq \Sigma$, let RS be the set of reachable states of M , δ' be δ restricted to $RS \times I$, and ν' be ν restricted to RS : $\delta'(q, a) = \delta(q, a)$, $\nu'(q) = \nu(q)$. Let $\Gamma' = \text{range}(\nu') \cup \{0, \omega\}$, where $\text{range}(\nu')$ denotes the range of ν' . The *joint network function Moore machine* M' of M_1 and M_2 can be defined by $M' = (RS, I, \Gamma', \delta', \nu', q_0)$.

Clearly a joint network function Moore machine satisfies the definition for a network function Moore machine.

Example 7: The joint network function Moore machine M of the power-saving and the load-balancing automata is omitted due to space. Only 13 out of the 42 states of $Q_1 \times Q_2$ are reachable, and included in the joint automaton.

VI. PROPERTY ANALYSIS AND CONFLICT DETECTION

We present analysis of the joint network function Moore machine, which enables detection of control conflicts as well as answers the many questions regarding the network's behavior. Our analysis focuses on three critical and closely related concepts: stable states of a Moore machine, safe operational region of a network, and conflict freeness of SDN apps.

A. Stable States of a Moore Machine

Intuitively, an SDN app enters a stable state, if it no longer attempts to modify the network state (i.e., the network has reached a desirable state from the app's perspective). This intuition can be formally defined using the automaton model. To do so, we extend the output function of a Moore machine to pairs of states and input sequences as follows. $\hat{\nu} : Q \times \Sigma^* \rightarrow \Gamma^*$ is defined by (1) $\hat{\nu}(q, \lambda) = \nu(q)$, and (2) $\hat{\nu}(q, wa) = \hat{\nu}(q, w)\nu(\hat{\delta}(q, wa))$ for all $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.

We can now formally define the stable states of a network function Moore machine as follows:

Definition 8 (Stable State): Let $q \in Q$ be a state of the network function Moore machine $M = (Q, \Sigma, \Gamma, \delta, \nu, q_0)$. q is a *stable state* of M iff the following hold for all $a \in \Sigma$:

- (1) $\delta(q', a) = q$ implies $\hat{\nu}(q, a^n) = 0^{n+1}$ for all integer $n \geq 0$, and
- (2) $q = q_0$ implies $\nu(q_0) = 0$.

Informally, two conditions need to be satisfied for a state to be a *stable state*: (1) on any input by which there is an incoming arc to this state, if a sequence of such input continues it will never land on any state that produces a non-null response, which could potentially change this input (i.e., a network state); and (2) the starting state must have the null output to be a stable state.

It is easy to see that this definition enforces any stable state be associated with the null response, as shown by the following theorem (proof is omitted due to lack of space).

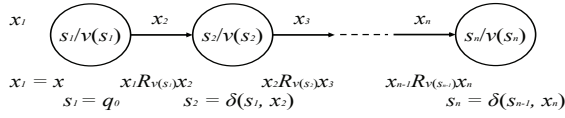


Fig. 4. An Example Sequence of Inputs and States for Analyzing the Safe Operational Region

Theorem 9: Let q be a stable state of the network function Moore machine $M = (Q, \Sigma, \Gamma, \delta, \nu, q_0)$. $\nu(q) = 0$.

Example 10: Applying Definition 8, it can be straightforwardly discovered that the constructed power-saving automaton M_1 has two stable states r_0 and r_3 ; the constructed load-balancing automaton M_2 has one stable state q_0 ; and the joint automaton M (Example 7) has two stable states (r_0, q_0) and (r_3, q_0) . We omit the proofs due to space constraint.

B. Safe Operational Region of a Network

Intuitively, a network becomes “stabilized” when none of the SDN apps running on top of it attempts to change its state; that is, all the SDN apps have entered a stable state as defined by Definition 8. Thus we consider a network state to be “safe”, if it can eventually (i.e., after a finite number of state changes) lead to a stabilized network, which guarantees no state oscillation. We term the set of all such safe states of a network to be the *safe operational region* of the network.

Note that a network’s state is already encapsulated in the stimuli (i.e., inputs) of a network function Moore machine. Hence we are able to formally define the concept of safe operational region of a network, as follows:

Definition 11 (Safe Operational Region): Given a network function Moore machine $M = (Q, \Sigma, \Gamma, \delta, \nu, q_0)$, a set of binary relations R_r over Σ for each $r \in \Gamma - \{\omega\}$ such that (1) $R_0 : \Sigma \rightarrow \Sigma$ is the identity function; and (2) $(r \neq 0, aR_rb)$ implies either $\delta(p, a) = q$, $\nu(q) = r$ for some $p, q \in Q$ or $\nu(q_0) = r$, and $x \in \Sigma$, x is in the *safe operational region* iff the alternating sequence of inputs and states $x_1, s_1, x_2, s_2, \dots, x_n, s_n$, in which $x = x_1$, $q_0 = s_1$, $x_{i-1}R_{\nu(s_{i-1})}x_i$, $s_i = \delta(s_{i-1}, x_i)$ for all $i \geq 2$, $\nu(s_i) \neq \omega$ for all $i \geq 1$, ends with some stable state s_n of M . The *safe operational region* of the network is the set of all such x ’s.

The set of binary relations R_r over I define how each output of an SDN app modifies the status of the network (notice that the null response does not modify the network status, as indicated by the identity function). Informally, a network state, as encapsulated in the input x , is in the safe operational region if and only if a stable state of the Moore machine will always be reached when starting from the initial state of the Moore machine with the input x . At that point the network state will no longer be changed by the running SDN app(s). Fig. 4 illustrates an example sequence of inputs and states that need to be examined from the initial state q_0 with a specific input (say x).

Example 12 (Safe Operational Region of Example Network): Applying Definition 11 on the joint automaton as defined in Example 7, it can be computed that the safe operational region of the example network (Fig. 1) includes H_wH_v , L_wZ_v , and Z_wZ_v . We omit the algorithm here due to lack of space.

C. Conflict Freeness of SDN Apps

An SDN network is free of conflict, if starting from any state it can eventually become stabilized; that is, every possible network state is in its safe operational region. We have the following formal definition:

Definition 13 (Conflict-Free SDN apps): Two or more SDN apps are *conflict-free* if and only if their joint network function Moore machine has the set of inputs identical to the safe operational region of the network.

Note that Definition 13 not only enables direct detection of potential conflict, but also precisely specifies under what network condition such conflict will arise.

Example 14 (Conflict in the Example Network): Applying Definition 13, we immediately see that the two SDN apps running on the example network are not conflict-free: this is because a subset of inputs to the joint network function Moore machine including H_wL_v , H_wZ_v , L_wH_v , L_wL_v , Z_wH_v , and Z_wL_v , is left out of the safe operational region.

VII. CONCLUSION AND FUTURE WORK

We presented a theoretical framework, based on rigorous software specification, for detecting conflicts caused by running multiple SDN apps in parallel. Our future work includes experimental validation and further investigation on the scalability and applicability of the framework.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation (NSF) under Grant CNS-1660569. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

- [1] D. Kreutz et al. “Software-Defined Networking: A Comprehensive Survey”. In: *Proc. of the IEEE* 103.1 (2015), pp. 14–76.
- [2] *Airheads Community SDN App Store*. <https://community.arubanetworks.com/t5/SDN-Apps/ct-p/SDN-Apps>.
- [3] D. Volpano, X. Sun, and G. Xie. “Towards Systematic Detection and Resolution of Network Control Conflicts”. In: *Proc. of ACM HotSDN*. Chicago, IL, 2014.
- [4] S. Prowell and J. Poore. “Sequence-based software specification of deterministic systems”. In: *Software: Practice and Experience* 28.3 (1998), pp. 329–344.
- [5] S. Prowell et al. *Cleanroom Software Engineering: Technology and Process*. Reading, MA: Addison-Wesley, 1999.
- [6] S. Prowell and J. Poore. “Foundations of sequence-based software specification”. In: *IEEE Transactions on Software Engineering* 29.5 (2003), pp. 417–429.
- [7] L. Lin, S. Prowell, and J. Poore. “An axiom system for sequence-based specification”. In: *Theoretical Computer Science* 411.2 (2010), pp. 360–376.
- [8] J. Mogul et al. “Corybantic: Towards the Modular Composition of SDN Control Programs”. In: *Proc. of ACM HotNets*. 2013.
- [9] A. AuYoung et al. “Democratic Resolution of Resource Conflicts Between SDN Control Programs”. In: *Proc. of ACM CoNext*. Sydney, Australia, 2014.
- [10] A. Bairley and G. Xie. “Orchestrating network control functions via comprehensive trade-off exploration”. In: *Proc. of IEEE NFV-SDN*. Palo Alto, CA, 2016.
- [11] A. Khurshid et al. “VeriFlow: Verifying Network-Wide Invariants in Real Time”. In: *Proc. of USENIX NSDI*. 2013.
- [12] P. Kazemian et al. “Real Time Network Policy Checking Using Header Space Analysis”. In: *Proc. of USENIX NSDI*. 2013.
- [13] *Software-Defined Networking: The New Norm for Networks*. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>.
- [14] P. Sun et al. “A Network-state Management Service”. In: *Proc. of ACM SIGCOMM*. Chicago, IL, 2014.
- [15] M. Al-Fares et al. “Hedera: Dynamic Flow Scheduling for Data Center Networks”. In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. San Jose, California, 2010.
- [16] S. Jain et al. “B4: Experience with a Globally-deployed Software Defined WAN”. In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. Hong Kong, China, 2013.
- [17] B. Heller et al. “ElasticTree: Saving Energy in Data Center Networks.” In: *Proceedings of USENIX NSDI*. 2010.

The Influence of God Class and Long Method in the Occurrence of Bugs in Two Open Source Software Projects: An Exploratory Study

Aloisio Sampaio Cairo [★]

Glauco de Figueiredo Carneiro [★]

Antônio Maria Pereira de Resende [✉]

Fernando Brito e Abreu [☆]

[★] Universidade Salvador (UNIFACS), Brazil

[✉] Universidade Federal de Lavras (UFLA), Brazil

[☆] Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Abstract

Context: Code smells are associated with poor design and programming style that often degrades code quality and hampers code comprehensibility and maintainability.

Goal: In this paper, we investigated to which extent classes affected by the God Class and Long Method code smells were more susceptible to the occurrence of software bugs.

Method: We conducted an exploratory study targeting two well known open source software projects, Apache Tomcat and Eclipse JDT Core Component. We applied correlation analysis in order to evaluate to which extent Long Method and God Class were related to the occurrence of bugs.

Results: We have found a significant correlation of Long Method and Commits and, on the other hand, a poor correlation of God Class and Commits in the two analyzed projects. Therefore, we expected that the higher the number of occurrences of Long Method, the higher the chances of more commits in a class that contains this method, which could result in the increase of occurrence of bugs.

Conclusion: Based on the results, we confirmed what other studies pointed out, regarding classes affected by Long Method being more bug-prone than others. In practice, we found evidence, from analyzed data, that the occurrence of Long Method implies more effort in maintenance tasks.

1 Introduction

Throughout releases, a software project usually includes new functionalities and fixes bugs that were found and considered relevant in previous releases. Since maintenance costs are the highest share in the development lifecycle [1], special attention should be dedicated to avoid bugs.

Martin Fowler, in his book on software refactoring [2], claimed that code smells can negatively impact software quality, namely on its understandability and reliability. For more than a decade after that, several authors have claimed

that the empirical evidence of such impact was still scarce [3, 4, 5] and, as such, Fowler's claim could not be considered as fully confirmed. Among the 22 code smells presented in Fowler's book, we selected *God Class* and *Long Method* to provide further evidence of their influence on software bugs occurrence. The former refers to classes that centralize the intelligence of the system, realizing too much work, when compared with the remainder classes, while the latter refers to methods with many statements and/or complex control flow structures [6].

To reach our stated research objective, we analyzed two projects, Apache Tomcat and Eclipse JDT Core Component. Tomcat is a "pure Java" HTTP web server environment in which Java code can run. JDT (Java Development Tools) Core Component is the Java infrastructure of Eclipse's Java IDE. We have chosen these two projects because they: (i) are open source; (ii) have a prominent popularity in the Java community; (iii) have been studied in the literature by several authors [7, 8, 9]; (iv) are related in the scope of this study.

Regarding the last reason, we found the relationship in their bug repository: 51 closed bugs from Apache Tomcat are related to the Eclipse project. These bugs were reported in classes affected by *God Class* or containing methods affected by *Long Method* in the following releases analyzed in this study: 9_0_9 of (Apache Tomcat) and Y20180725-2200 (Eclipse JDT Core). Moreover, we have also noticed that 36 bugs out of those 51 were reported in classes simultaneously affected by both code smells in those releases. Hence, we considered that the influence of the aforementioned code smells in the selected software projects deserved further analysis.

The remainder of this paper is laid out as follows. Section 2 overviews previous related research work. The experimental design of our study is presented in Section 3 and Section 4 discusses its results. Finally, in Section 5, we present concluding remarks and future work.

2 Related Work

Lehman’s second law of software evolution [10] argues that as a project grows, software complexity increases as well. Constantly changing code is the context in which code smells find favorable conditions to arise. Studies have reported that code smells usually affect a software entity, such as class or method, after a change executed to fix a bug [11]. To tackle this problem, several approaches have been proposed to support the identification of code smells, as described in secondary studies such as the ones conducted by Rasool and Arshad [12] and Fernandes et al. [13].

According to [14], code smells are indicators of poor source code quality and hamper its maintenance and reuse. The influence of the *God Class* and *Long Method* code smells on software bugs occurrence has been studied by several authors. Li and colleagues [15] reported the influence of *Shotgun Surgery*, *God Class*, and *God Method* code smells on bugs based on the analysis of post-release system evolution. They confirmed that some smells were positively associated with the class error probability in three error-severity levels (High, Medium, and Low), usually applied to classify issues. In [16], the authors unveiled the relationship among *Shotgun Surgery*, *God Class* and *God Methods* code smells and the occurrence of bugs in three Eclipse releases (3.0, 2.1 and 2.0). Olbrich and colleagues [17] presented evidence that instances of *God Class* and *Brain Class* suffered more frequent changes and contained more defects than classes not affected by those smells.

Later, Nascimento and Sant’Anna [18] analyzed five software projects (Apache Ant, Apache Jmeter, Apache Lenya, Apache Tomcat and Apache Xerces) to conclude that classes affected by the code smells *Data Class*, *Data Clumps*, *Feature Envy*, *God Class*, *Message Chain*, *Schizophrenic Class* and *Tradition Breaker* had a higher likelihood to introduce bugs than other classes from the same project. They further reported *God Class* as the smell with the greatest number of related bugs in the analyzed projects with a percentage of 20%, followed by *Feature Envy* with a percentage close to 15%, *Schizophrenic Class* with 9% and *Message Chains* with 7%. In addition, Palomba and colleagues [8] claimed that classes with code smells tend to be more change- and fault-prone than other classes and that this is even more noticeable when the same class is affected by multiple smells.

A recent systematic literature review conducted by Cairo and colleagues [19] claims that there is evidence in the literature showing that software entities, such as classes affected by code smells, tend to be more prone to change and failure than other classes.

3 The Experimental Design

This section describes the characteristics and steps applied in this exploratory study. We adopted the Goal-

Question-Metric (GQM) approach [20], as described in Table 1, to derive the following **Research Question (RQ)**: To which extent *God Class* and *Long Method* code smells influence the occurrence of bugs in Apache Tomcat and Eclipse’s JDT Core Component projects?

Table 1: Research goal based on the GQM approach

Analyze	the <i>God Class</i> and <i>Long Method</i> code smells
for the purpose of	evaluating their influence
with respect to	the occurrence of bugs
from the viewpoint of	maintainers and developers
in the context of	Apache Tomcat and Eclipse’s JDT Core Component open source software projects

To reduce the collection time and the frequent errors that arise during code smells manual detection [21], we used the PMD static code analyzer. We performed the six steps represented in Figure 1 that will be further detailed in the following paragraphs, using a common template (data input, step description, performing the step, data output).

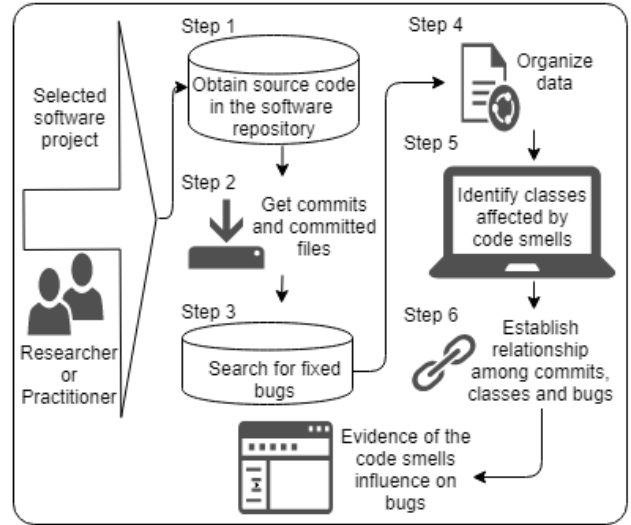


Figure 1: Proposed approach to analyze the influence of code smells on bugs occurrence in a software project

3.1 Step 1 - Obtain source code

Data input: URL of selected software projects repository.

Step description: We cloned the sources of each selected software project using Git Bash, a Windows application which provides a Git command line shell.

Performing the step: We executed the `git clone` command.

Data output: Local repository containing the source code of selected software projects.

3.2 Step 2 - Get commits and corresponding files

Data input: GitHub features that allow the collection of data related to commits performed in a software project.

Step description: We executed the command lines 1 and 2 presented below using Git Bash. Besides commits, we also collected the files involved in each commit.

Performing the step: Execute the commands as follows:

(1) `git log --pretty=format:"%H %h %an %ad %s" --grep="bug" > log-commit.txt` (stores the names of all files involved in each commit in the named file)

(2) `git show hash-numbers --pretty=format: "%H" --stat > log-arquivos.txt` (stores all commit hashes in the named file)

3.3 Step 3 - Search for fixed bugs

Data input: Bugzilla features that allow the collection of data related to bugs of a specific software project.

Step description: Identify all bugs marked as *closed* for all analyzed software projects.

Performing the step: We collected all closed bugs for Apache Tomcat until August 8th (2018) and for Eclipse until September 9th (2018). We used the Bugzilla's REST API to obtain data from the projects' repositories.

Data output: Files containing information related to bugs in JSON format from the analyzed software projects.

3.4 Step 4 - Organize data

Data input: Data about commits and bugs for each project.

Step description: We tabulated raw data to prepare them for statistical analysis.

Performing the step: All data was inserted in a database and we applied queries to get it organized in table format. After that, we exported them to a spreadsheet for applying statistical analysis. An automatic routine was implemented to capture the ID from Bugzilla repositories. Only Java files were considered and all others (e.g. configuration files) were deleted, since only the former were relevant for code smells detection. We also did not consider commits in files not related to code smells. Some long absolute paths to Java classes were incomplete, such as, for instance in `"/.../apache/catalina/connector/CoyoteAdapter.java"`. In those cases, we searched manually for the file and completed the correct absolute path string, being careful in cases of multiple files with the same name, located in different packages.

Data output: Commit data, file involved in each commit, and bugs of each project.

3.5 Step 5 - Identify classes affected by code smells

Data input: Source code of projects cloned in step 1.

Step description: We used the PMD tool to collect information about code smells.

Performing the step: The PMD tool was applied in each project to detect *God Class* and *Long Method* occurrences.

Data output: A table containing all software project classes

affected by *God Class* and *Long Method*, and the number of occurrences of each code smell, for each class.

3.6 Step 6 - Evidence of code smells influence on bugs

Data input: Each line of tabulated data contains software project name, version, absolute path of Java class, class name, occurrences of *Long Method*, occurrences of *God Class* and total amount of commits involving both projects.

Step description: We performed statistical analysis on available data to check the intensity of correlations among code smells and bugs.

Performing the step: Since the data was not normally distributed, we used the Spearman non-parametric correlation coefficient to check the correlation among the occurrences of each code smell and the amount of commits. We grouped the commits in ranges to check the correlation. *Data output:* Intensity of correlation among code smells occurrences and amount of bugs, represented by commits.

4 The Study Results

4.1 Distribution adherence testing

All data used in this analysis is available in a public Github repository¹. The total number of classes analyzed with at least one commit are 1049 and all those classes have at least one code smell. The first step of a statistical correlation analysis is to verify whether the data is normally distributed. This verification allows the selection of the most suitable statistical technique to be applied. We used the EasyFit tool² to perform the aforementioned verification. The results pointed out that the occurrences of *Long Method* and *God Class* have Poisson Distribution and the amount of commits has Geometric distribution. Since none of them has a normal distribution, that required the application of a non-parametric correlation coefficient such as Spearman's.

4.2 Correlation analysis by commit ranges

The main research question stated in Section 3 aimed at devising to which extent *God Class* and *Long Method* code smells influence (or have impact in) the occurrence of bugs in two open source projects. In this study we only considered bugs that were fixed and took as surrogate of those bugs the number of file (class) commits they cause. Therefore, the research question originated two sub-questions, one per each code smell, where we want to check if there is a statistical significant association among the variables that allow us to not discard that the aforementioned impact exists. In other words, the sub-questions are:

a) *Is there a correlation between the occurrences of Long Method in a class and the amount of commits in that class?*

¹<https://github.com/SEKE2019CodeSmell/SEKE2019EvidenceCodeSmells>

²<http://www.mathwave.com/>

b) Is there a correlation between the occurrences of *God Class* in a class and the amount of commits in that class?

To answer those questions, while eliminating some random deviations that might have occurred in collected data, we grouped the analyzed classes by consecutive ranges of 3 commits. Therefore, the first range contains classes with commits within the $[0, 3[$ range, the second in the $[3, 6[$ range, and so forth, as represented in the first two columns in Tables 2 and 3 for the Apache Tomcat and Eclipse JDT Core Component, respectively. For instance, the values of 130 and 279 on the first line of column II of the same tables represent the total number of classes with 0, 1 or 2 commits (range defined in column I) in the two analyzed software projects. We have also normalized the amount of class commits, occurrences of *Long Method* and *God Class*, by calculating their average value per class. The column labels of Tables 2 and 3 are as follows:

(I) Commit range (\geq and $<$, respectively)

(II) Class total

(III) Commits Sum

(IV) Long Method Total (occurrences)

(V) God Class Total (occurrences)

(VI) Code Smells Total (occurrences)

(VII) Average Commits Per Class

(VIII) Long Method Average (occurrences) Per Class

(IX) God Class Average (occurrences) Per Class

The highlighted lines in Tables 2 and 3 represent the data that we considered as relevant. The classes with commits above the 57-60 range were clear outliers and were ignored in the following study. To confirm this claim, we generated dispersion charts (Figures 2 and 3) to visually check the distribution of data in the ranges. The *Long Method* chart provides some evidence of correlation, because the points are around the tendency line and increasing, meaning a positive correlation. Furthermore, it is possible to obtain a curve with even better fit (as measured by the R squared coefficient of determination), what suggests that the effect of this code smell on commits may be non-linear. As for the *God Class* the chart does not provide evidence of correlation. After determining the valid data, without irrelevant lines and outliers, we calculated the Spearman correlations. The results are presented in Table 4.

To interpret the values of the coefficients in Table 4, we used the ordinal intensity scale presented in Table 5 (from [22]). Therefore, based on the obtained values of the Spearman correlation coefficient in Table 4, we can state that there is a *very high positive correlation* between *Long Method* Average and Commit Average. Consequently, that result allow us to raise the hypothesis that the number of commits in a class may be impacted by the occurrences of the *Long Method* code smell. The same cannot be claimed for the *God Class* code smell, since only a moderate positive correlation was observed. For the Eclipse project, we

Table 2: Classes grouped by commit ranges (APACHE TOMCAT)

(I)		(II)	(III)	(IV)	(V)	(VI)	(VII)	(VIII)	(IX)
0	3	130	105	47	109	156	0,81	0,36	0,84
3	6	71	282	27	64	91	3,97	0,38	0,90
6	9	47	330	16	42	58	7,02	0,34	0,89
9	12	23	237	7	24	31	10,30	0,30	1,04
12	15	22	283	14	22	36	12,86	0,64	1,00
15	18	14	226	11	14	25	16,14	0,79	1,00
18	21	12	229	6	11	17	19,08	0,50	0,92
21	24	8	177	9	9	18	22,13	1,13	1,13
24	27	7	171	9	7	16	24,43	1,29	1,00
27	30	1	27	1	1	2	27,00	1,00	1,00
30	33	4	122	3	4	7	30,50	0,75	1,00
33	36	1	35	4	1	5	35,00	4,00	1,00
36	39	2	74	4	2	6	37,00	2,00	1,00
39	42	1	39	3	1	4	39,00	3,00	1,00
42	45	1	42	4	1	5	42,00	4,00	1,00
45	48	1	46	3	1	4	46,00	3,00	1,00
48	51	0	0	0	0	0	0,00	0,00	0,00
51	54	1	51	8	1	9	51,00	8,00	1,00
54	57	0	0	0	0	0	0,00	0,00	0,00
57	60	0	0	0	0	0	0,00	0,00	0,00
60	63	0	0	0	0	0	0,00	0,00	0,00
63	66	0	0	0	0	0	0,00	0,00	0,00
66	69	0	0	0	0	0	0,00	0,00	0,00
69	72	0	0	0	0	0	0,00	0,00	0,00
72	75	0	0	0	0	0	0,00	0,00	0,00
75	78	1	75	6	1	7	75,00	6,00	1,00
78	81	0	0	0	0	0	0,00	0,00	0,00
...
90	93	0	0	0	0	0	0,00	0,00	0,00

have a high positive correlation for *Long Method* Average and Commit Average, and for *God Class* Average and Commit Average. Those correlations in the Eclipse project are lower than in the Tomcat project, even though both projects presented a significant correlation. When we consider both projects together, the results keep in-between, as expected.

4.3 Answering the research questions

As discussed in section 4.2, our main research question was split in two, each regarding the association between the occurrences of one code smell and the corresponding commits. Considering the observed high correlation, we cannot refute the hypothesis that the number of occurrences of *Long Method* may have a positive impact on the amount of commits. As for the *God Class* code smell, due to the observed moderate to low positive correlation, we have to say it the other way round: we cannot sustain the hypothesis that the number of occurrences of *Long Method* may have an impact on the amount of commits.

4.4 Threats to Validity

We are aware of the limitations of correlational analyses such as the used in this study. For this reason, we dubbed this paper as an *exploratory study*. The evidence raised basically allows to stand or discard hypotheses that must be tested, preferably with controlled experiments. The conclu-

Table 3: Classes grouped by commit ranges (ECLIPSE JDT CORE)

(I)	(II)	(III)	(IV)	(V)	(VI)	(VII)	(VIII)	(IX)
0	3	279	227	199	109	308	0,81	0,71
3	6	142	541	115	64	179	3,81	0,81
6	9	80	548	81	42	123	6,85	1,01
9	12	56	554	93	24	117	9,89	1,66
12	15	41	521	91	22	113	12,71	2,22
15	18	23	367	60	14	74	15,96	2,61
18	21	17	325	43	11	54	19,12	2,53
21	24	12	260	21	9	30	21,67	1,75
24	27	14	344	84	7	91	24,57	6,00
27	30	15	418	100	1	101	27,87	6,67
30	33	4	124	14	4	18	31,00	3,50
33	36	2	66	2	1	3	33,00	1,00
36	39	1	37	1	2	3	37,00	1,00
39	42	3	120	7	1	8	40,00	2,33
42	45	0	0	0	1	1	0,00	0,00
45	48	4	184	13	1	14	46,00	3,25
48	51	0	0	0	0	0	0,00	0,00
51	54	1	52	6	1	7	52,00	6,00
54	57	0	0	0	0	0	0,00	0,00
57	60	1	57	3	0	3	57,00	3,00
60	63	0	0	0	0	0	0,00	0,00
63	66	0	0	0	0	0	0,00	0,00
66	69	0	0	0	0	0	0,00	0,00
69	72	0	0	0	0	0	0,00	0,00
72	75	0	0	0	0	0	0,00	0,00
75	78	0	0	0	0	0	1	1
78	81	1	79	11	0	11	79,00	11,00
81	84	0	0	0	0	0	0,00	0,00
84	87	0	0	0	0	0	0,00	0,00
87	90	1	89	0	0	0	89	0,00
90	93	0	0	0	0	0	0,00	0,00

sion validity is related to the ability to draw significant correct conclusions. For this reason, we applied the distribution adherence testing and the correlation analysis by commit ranges based on dispersion charts to support the analysis. We discussed the internal validity in terms of threats to the design of the study. For this reason, we managed to carefully plan the study and the variables to consider in the analysis as described in the six steps presented in Section 3. The use of the PMD tool for collecting code smells can be also a potential threat, since several authors have pointed out the inherent subjectivity in the code smells definition [23] can lead to unmatched detection, when using different tools. We expect to mitigate this last problem through our ongoing research work on the Crowdsourcing approach [24]. The external validity represents the possibility of generalizing the findings of this study. We identified external validity threats in our results, since the analysis was restricted to data from two open source software project systems, implemented in one programming language and focusing on two code smells from the catalogue published in [2]. However, the fact that these two open source software projects have an active community is an evidence of the representative-

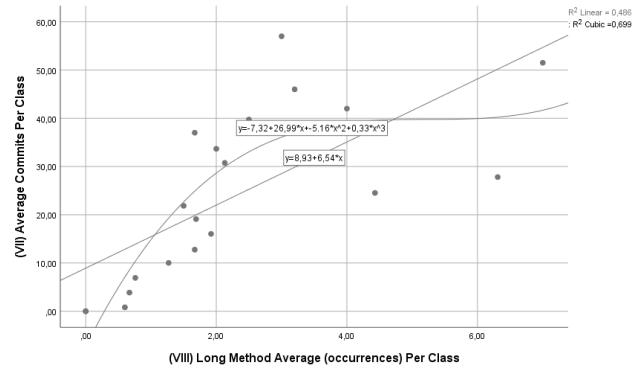


Figure 2: Dispersion Chart between Average Commits and Long Method Average occurrences (both projects)

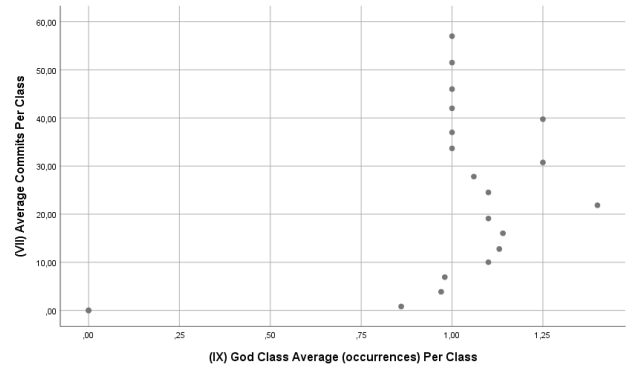


Figure 3: Dispersion Chart between Average Commits and God Class Average occurrences (both projects)

ness of these projects to support the conclusions drawn is this study for the analyzed code smells implemented in the Java language.

5 Conclusions

In this paper, we conducted an exploratory study upon two open source projects (Apache Tomcat and Eclipse JDT Core Component), in order to check if the occurrences of the *Long Method* and *God Class* code smells were associated to software defects (bugs). The PMD tool was applied to gather those code smells from all classes of those projects where code fixes occurred. The REST API of the Bugzilla tool was used to identify which were the files affected by code fixes (i.e. those where commits occurred during code fixing), by collecting data related to all closed bugs of those projects. We applied the Spearman correlation coefficient to evaluate the association between average values (per class) of a) occurrences of *Long Method* and observed commits; b) occurrences of *God Classes* and observed commits; and c) occurrences of both code smells and observed commits. We obtained a strong correlation of *Long Method* and Commits and revealed a poor correlation of *God Class*

Table 4: Spearman correlations (code smells vs commit ranges)

Project	Long Method average vs Commit average	God Class average vs Commit average
Tomcat	93%	52%
Eclipse	72%	30%
Both	85%	33%

Table 5: Interpreting the Correlation Coefficient [22]

Correlation	Interpretation of correlation
90% to 100%	Very high positive (negative)
70% to 90%	High positive (negative)
50% to 70%	Moderate positive (negative)
30% to 50%	Low positive (negative)
0% to 30%	Negligible correlation

and Commits, regarding both projects analyzed. Therefore, the higher is the number of *Long Method*, the higher is expected to be the number of commits in a class, although the effect is probably non-linear. In practice, we got strong evidence in those datasets that the occurrence of the *Long Method* may make the maintenance process harder, so we should avoid it through refactoring operations.

References

- [1] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT professional*, vol. 2, no. 3, pp. 17–23, 2000.
- [2] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.
- [3] S. M. Olbrich, D. S. Cruzes, and D. I. K. Sjöberg, "Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems," in *2010 IEEE International Conference on Software Maintenance*, Sep. 2010, pp. 1–10.
- [4] A. Yamashita and L. Moonen, "To what extent can maintenance problems be predicted by code smell detection?—an empirical study," *Information and Software Technology*, vol. 55, no. 12, pp. 2223–2242, 2013.
- [5] D. I. K. Sjöberg, A. Yamashita, B. C. D. Anda, A. Mockus, and T. Dybå, "Quantifying the effect of code smells on maintenance effort," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1144–1156, Aug 2013.
- [6] M. Lanza and R. Marinescu, *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media, 2007.
- [7] H. Aman, "An empirical analysis on fault-proneness of well-commented modules," in *Empirical Software Engineering in Practice (IWESEP), 2012 Fourth International Workshop on*. IEEE, 2012, pp. 3–9.
- [8] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia, "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," *Empirical Software Engineering*, pp. 1–34, 2017.
- [9] A. Saboury, P. Musavi, F. Khomh, and G. Antoniol, "An empirical study of code smells in javascript projects," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 2017, pp. 294–305.
- [10] M. M. Lehman, "Laws of software evolution revisited," in *European Workshop on Software Process Technology*. Springer, 1996, pp. 108–124.
- [11] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and D. Poshyvanyk, "When and why your code starts to smell bad," in *Proceedings of the 37th International Conference on Software Engineering—Volume 1*. IEEE Press, 2015, pp. 403–414.
- [12] G. Rasool and Z. Arshad, "A review of code smell mining techniques," *Journal of Software: Evolution and Process*, vol. 27, no. 11, pp. 867–895, 2015.
- [13] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo, "A review-based comparative study of bad smell detection tools," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2016, p. 18.
- [14] M. Mika, J. Vanhanen, C. Lassenius et al., "A taxonomy and an initial empirical study of bad smells in code," in *null*. IEEE, 2003, p. 381.
- [15] W. Li and R. Shatnawi, "An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution," *Journal of systems and software*, vol. 80, no. 7, pp. 1120–1128, 2007.
- [16] F. Khomh, M. Di Penta, and Y.-G. Gueheneuc, "An exploratory study of the impact of code smells on software change-proneness," in *Reverse Engineering, 2009. WCRE'09. 16th Working Conference on*. IEEE, 2009, pp. 75–84.
- [17] S. M. Olbrich, D. S. Cruzes, and D. I. Sjöberg, "Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–10.
- [18] R. Nascimento and C. Sant'Anna, "Investigating the relationship between bad smells and bugs in software systems," in *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse*. ACM, 2017, p. 4.
- [19] A. Cairo, G. Carneiro, and M. Monteiro, "The impact of code smells on software bugs: A systematic literature review," *Information*, vol. 9, no. 11, p. 273, 2018.
- [20] V. Basili, G. Caldiera, and H. Rombach, "Goal question metric approach paradigm," pp. 528–532, 1994.
- [21] N. Moha, Y.-G. Gueheneuc, A.-F. Duchien et al., "Decor: A method for the specification and detection of code and design smells," *IEEE Transactions on Software Engineering (TSE)*, vol. 36, no. 1, pp. 20–36, 2010.
- [22] D. E. Hinkel, W. Wiersma, and S. G. Jurs, *Applied statistics for the behavioral sciences*, 5th ed. Houghton Mifflin Company, 2003.
- [23] S. Bryton, F. Brito e Abreu, and M. Monteiro, "Reducing subjectivity in code smells detection: Experimenting with the long method," in *2010 Seventh International Conference on the Quality of Information and Communications Technology*. IEEE, 2010, pp. 337–342.
- [24] J. P. dos Reis, F. Brito e Abreu, and G. Carneiro, "Code smells detection 2.0: Crowdsampling and visualization," in *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2017, pp. 1–4.

Feature Evaluation for Automatic Bug Report Summarization

Akalanka Galappaththi

Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Canada
a.galappaththi@uleth.ca

John Anvik

Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Canada
john.anvik@uleth.ca

Abstract—Bug reports can be lengthy due to long descriptions and long conversation threads. Automatic summarization of the text in a bug report can reduce the time spent by software project members on understanding the content of a bug report. Our work further examines Rastkar et al.'s use of a logistic regression model to determine which sentences from the text of a bug report should be extracted for creating a summary. Using their publicly available bug report corpus, which contains manually annotated bug reports, we examined two aspects regarding the features used by the model. First, we examined how much of a reduction occurs in the precision and recall if some of the more complex features are not used. Second, we examined how the use of different feature combinations affects the precision and recall of the models. We found that the absence of some of the complex features resulted in a modest decrease in precision and recall, and confirmed that some features, such as sentence length, were the most significant features for bug report summarization.

Index Terms—Bug reports, text summarization, software engineering, natural language processing

I. INTRODUCTION

Bug reports¹ are useful software project artifacts that contain information about problems occurring in the past, discussion of possible or implemented solutions, and who contributed to these solutions. Bug reports contain a variety of textual information including a title, description, and comments [1]. The number of sentences in a bug report can vary widely, with some bug reports containing relatively few sentences (around 20) and some having over 50 sentences.

Bug report triage is an important software maintenance task where a project member examines each bug report and makes decisions about how the report will be handled [1]. Examples of bug report triage decisions include determining the quality, priority or severity of the report, assignment to a developer, validation of the report, and identification of duplicate reports. As software projects receive many bug reports each day [2], to effectively triage a new bug report, the person performing triage, called a *triager*, needs to understand the content of the new bug report, as well as previous bug reports. If a bug report

is long (i.e. a long description and/or many comments), the triager needs to spend a significant amount of time reading and understanding the content. Presenting a summarized version of a bug report has been proposed to reduce the time taken by triagers in examining new or existing bug reports and has been shown to reduce the time for understanding the content without necessarily reducing the meaning of the original bug report [3], [4]. By reading the summarized version, a triager could more quickly triage a bug report, such as determining whether it is a duplicate of an existing bug report or whether it needs to be fixed. Bug report summarization can also benefit developers, as they could quickly get an overview of how a particular or related problem was handled in the past.

As the content of a bug report is primarily free-form text, natural language processing techniques, specifically those for text summarization, are used [3], [5]. There are two types of text summarization techniques: abstractive and extractive. Abstractive summarization requires understanding the content of the document and paraphrasing the content in such a way that the meaning is preserved. Extractive summarization selects sentences from the document that reflect the overall meaning of the document [6]. Text summarization uses both statistical and linguistic techniques to find the meaning and importance of a text.

Rastkar et al. [3] presented an extractive approach to bug report summarization that appears to work well. Their approach selects sentences to be extracted from the bug report based on a variety of features and a linear regression model. However, some of the features used in their approach are non-standard (e.g. clue word score), and extracting those features is complex. In this work, we investigate whether reasonable extractive summaries of bug reports can be created without the use of the complex features, and how the different features contribute to the creation of a reasonable extractive summary.

We begin by presenting an overview of previous bug report summarization work. We then present an overview of the data set and the extractive bug report summarization technique used in our investigation. Next, we describe how the summarizer was evaluated and the results of our investigation. We conclude the paper with a discussion of our findings.

¹We use the general term 'bug report' to refer to any software project artifact which is used for tracking project work such as feature requests, change requests, and tasks descriptions.

II. RELATED WORK

We present in this section an overview of previous work related to bug report summarization.

Murray and Carenini [7] proposed an approach for extracting four types of features to summarize email and meeting conversation data. The extracted features were related to the length and the structure of the sentences, the lexical weights based on the conditional probabilities of each word's appearance, and the participants of the conversation.

Rastkar et al. [3], [8] observed that comments in bug reports were similar to the conversation structure found in emails and meeting transcripts. Therefore they used the same feature extraction techniques as Murray to train a logistic regression model to classify sentences as being included in an extractive summary or not. They created an annotated bug report corpus for their investigation².

Lotufo, Malik, and Czarnecki [4] conducted a study of automatic, unsupervised bug report summarization that took a different approach than Rastkar et al.. Instead of using gold standard summaries, their study considered the similarity between a sentence and the bug report title, the similarity between two sentences, and the use of a heuristic to measure the agreement of two sentences. These characteristics were used to compare the quality of generated summaries. This approach led to a more generalized approach to evaluating bug report summaries.

Mani et al. [5] also proposed an unsupervised approach to text summarization. Instead of using the gold standard summaries, they used well known general purpose textual summarizers to create bug report summaries. However, they found that these summarizers only worked well when there was no noise present in the bug reports. To reduce the noise, they used heuristics to categorize text as being either a question, an investigative sentence or a code snippet.

III. BUG REPORT SUMMARIZATION

As previously stated, the goal of our study is to investigate the use of different features for extractive summarization of bug reports as presented by Rastkar et al. [3]. They categorized the 24 features into four sets: sentence length, lexical features, structural features and features related to the participants of the conversation in the bug report. As Rastkar et al. found that the F-score statistics of the structural and participant-related features have low variability, we focused our study on the length and lexical features. We also chose to remove the complex features of clue-word-score and those related to sentence entropy to investigate how extractive summaries selected without the use of these features compare to manually created summaries. In short, we investigated the use of two sentence length features, six conditional probability scores and four cosine similarity scores. The details for these features are explained in the following two sections.

²<https://www.cs.ubc.ca/cs-research/software-practicelab/projects/summarizing-software-artifacts> verified 09/12/2018

A. Data Source

We used the bug report corpus created by Rastkar et al. [3] in our investigation. The bug report corpus consists of thirty-six (36) bug reports extracted from four open source software projects: Mozilla³, KDE⁴, Eclipse⁵ and Gnome⁶. Each bug report in the corpus has a title which indicates from which software project it comes. Each bug report's comments are given in a format such as that of two or more people taking turns when having a conversation. Therefore, each comment is considered a *turn* and each turn has the participant's name, the time when the conversation started, and the text of the individual sentences from the entire comment. Each bug report has two or more people participating in the conversation. Bug reports are stored in an XML format, each identified by a unique number.

The corpus also contains annotations made by three different annotators for each comment indicating whether the sentence should be included in an extractive summary. Following the procedure given by Rastkar et al. [3], we created gold standard summaries (GSS) for each bug report by including a sentence in the extractive summary if at least two annotators indicated it should be included in the bug report summary.

B. Extracted Features

1) *Sentence Length*: The two sentence length features extracted from each bug report comment are SLEN and SLEN2. SLEN is the length of a sentence normalized by the length of the longest sentence in all of the comments in the bug report. SLEN2 is the length of a sentence normalized by the length of the longest sentence in the specific bug report comment.

2) *Lexical*: In our investigation, ten lexical features were extracted from each sentence in a bug report comment. These lexical features are based on two different conditional probabilities known as S_{prob} (for sentence probability) and T_{prob} (for turn probability).

Equation 1 defines S_{prob} . Given a term t by a person making a comment, S_{prob} is calculated by finding the maximum probability of that term's appearance in sentences from all of the comments S . For example, assume there are three commenters for a bug report: A , B and C . If A used the word w_1 seven times, B used w_1 twice and C used w_1 once, then the maximum probability of w_1 is 0.7, and all instances of w_1 in the bug report receive 0.7 as their S_{prob} weight.

$$S_{prob}(t) = P(S|t) \quad (1)$$

Equation 2 defines T_{prob} . Given a term t by a person making a comment, T_{prob} is calculated by finding the maximum probability of that term's appearance in a comment T . For example, if a bug report has five comments and the word w_2 appears eight times in one comment, twice in another comment, and in no other comments, then $T_{prob}(w_2)$ is 0.8.

³bugzilla.mozilla.org, verified 09/12/2018

⁴bugs.kde.org, verified 09/12/2018

⁵bugs.eclipse.org/bugs, verified 09/12/2018

⁶bugzilla.gnome.org, verified 96/12/2018

$$T_{prob}(t) = P(T|t) \quad (2)$$

Using each conditional probability weight we calculated three sentence level conditional probability features for the sum of weights, maximum weight and mean of weights. For S_{prob} the features calculated were named as SMS, MXS and MNS for sum, max and mean respectively. Similarly, for T_{prob} the names were SMT, MXT and MNT.

We calculated four cosine similarity scores for each sentence using S_{prob} and T_{prob} as the word encoding. COS1 and COS2 represent the sentence wise cosine similarity using S_{prob} and T_{prob} respectively. CENT1 and CENT2 represent the similarity of a sentence to the entire conversation using S_{prob} and T_{prob} respectively.

C. Bug Report Summary Creation

A logistic regression model was used to train a recommender which classifies each sentence from the bug report comments as appearing or not appearing in the extractive summary. The Python `sklearn` package was used to implement the sentence classifier. Various logistic regression models were trained to test the effect of the individual feature types and the combined effect of the length and lexical features regarding the performance of the classifier.

Following the procedure outlined by Rastkar et al., a bug report summary was created by selecting the sentences with the highest probability of being included in the extractive summary according to the classifier until the word count of the constructed summary reached 25% of the original bug report's word count.

IV. EVALUATION

To compare our model with Rastkar et al. [3] we trained a logistic regression model using our twelve selected features. We also investigated the effect of using only one of the two feature categories and different feature combinations. To investigate the effect of different feature combinations, we paired similar features into six groups (length, sum, max, mean, cos, cent) and created logistic regression models with combinations of the different groups. For example, choosing one group at a time creates six models (i.e. $\binom{6}{1}$) and choosing two groups at a time creates fifteen models (i.e. $\binom{6}{2}$). We created a total of sixty-two (62) models by choosing all combinations of the six groups (i.e. $\binom{6}{1}$ to $\binom{6}{5}$).

We used the metrics of precision, recall and F-score to evaluate the different models.

Precision measures how many of sentences in the extractive summary were correct. It was calculated as the total number of sentences correctly classified as being in the extractive summary divided by the total number of sentences in the extractive summary.

$$Precision = \frac{\# \text{ of sentences correctly selected}}{\text{total \# sentences in the summary}} \quad (3)$$

Recall measures how close the generated summary is to the GSS summary. It was calculated as the total number of sentences correctly classified as being in the extractive summary divided by the total number of sentences which appeared in the GSS summary.

$$Recall = \frac{\# \text{ of sentences correctly selected}}{\text{total \# sentences in GSS summary}} \quad (4)$$

The F-score value is the harmonic mean of precision and recall.

$$F - score = \frac{2 \times precision \times recall}{precision + recall} \quad (5)$$

As we used a leave-one-out cross-validation approach to evaluate each model, we received thirty-six (36) data points for the precision and recall of each model. We applied a post-hock pairwise statistical test⁷ to determine if there was any statistically significant variation between the models.

V. RESULTS

In this section, we present a comparison of our extractive summarization approach with Rastkar et al.'s, the results of our investigation of using only sentence or lexical features, and the results of our investigation of combinations of our selected features.

A. Comparison to the Previous Approach

When we compare the results of our logistic regression model with a 25% threshold with that of Rastkar et al.'s, we found that our model has an average precision and recall that is less than theirs (Table I). However, this decrease is to be expected, given that fewer features are used in our model.

Our comparison with Rastkar et al.'s approach prompted us to further examine the contribution to extractive summarization of the individual features and their combinations.

TABLE I
COMPARISON OF OUR RESULTS WITH RASTKAR'S RESULTS

	Our model	Rastkar's model
Precision	44%	57%
Recall	24%	35%
F-Score	29%	40%

B. Feature Categories

We examined the use of only sentence length features and only lexical features for creating an extractive bug report summarizer.

It was found that when using only the sentence length features (SLEN and SLEN2), the classifier had an average precision of 60% and the recall was found to be very low at 18%.

Using only the lexical features (SMS, MXS, MNS, SMT, MXT, MNT, COS1, COS2, CENT1, CENT2), we found that

⁷Statistical analysis was conducted using R 3.5.1.

the recall improved to 22%, but the precision declined to 42%. The model which combined both the length and lexical features had a precision of 44% and a recall of 24%. As shown in Table II, the F-score was found to improve from 27% to 30% after adding the lexical features.

TABLE II
AVERAGE PRECISION, RECALL AND F-SCORE VALUE OF LEAVE-ONE-OUT
CROSS VALIDATION FOR CLASSIFIERS WITH 25%.

Feature set	Precision	Recall	F-Score
Length	60%	18%	27%
Lexical	42%	22%	27%
All	44%	24%	29%

C. Feature Combinations

To understand the combined contribution of the features and to discover if there was a difference in the performance if we chose different combinations of features, we created sixty-two (62) different models by choosing different group combinations.

When testing for significance in the differences between combinations of features, only the sentence length features in $\binom{6}{1}$ were found to be significant. The p-values for these comparisons are shown in Table III.

TABLE III
SIGNIFICANCE OF DIFFERENCE IN THE PRECISION FOR $\binom{6}{1}$.

Feature combinations	p-value
SLEN-SLEN2 vs. CENT1-CENT2	0.0062577
SLEN-SLEN2 vs. COS1-COS2	0.0004581
SLEN-SLEN2 vs. MNS-MNT	0.0013389

VI. DISCUSSION AND CONCLUSION

In our study, we focused on two types of features: sentence length and lexical features. According to Rastkar et al. [3] SLEN was one of the most important features when training an extractive bug report summarizer because of its high variability. Two lexical features SSM and TSM and the sentence length feature SLEN2 were also considered important. However, they also felt that all 24 features contributed to the overall performance of their model regardless of some of the features having less variability. Our results seem to corroborate this belief with our statistical analysis for different feature combinations, finding no statistically significant difference in performance when adding or removing features.

In training the logistic regression model with the sentence length and lexical features, we found that lengthy sentences often contained useful information needed to create a good summary. This is consistent with the literature which states that the sentence length is used to eliminate short sentences from the summary that do not contain useful information, such as author names or code extractions [9]–[11]. However, we also found that only using the length of the sentence was not enough to capture all of the useful sentences, as

selecting longer sentences resulted in quickly reaching the word percentage threshold. Our results for the use of lexical features also show that the recall is higher when the length is not considered, as the classifier selects more of the shorter sentences thereby increasing the recall.

Finally, as we are taking a supervised learning approach to summary creation, the results are sensitive to the summaries found in the GSS. When comparing the word count of the summaries in the GSS for each bug report, we found that some summaries had a word count of more than 50% of the original bug report's word count. As we used a word count threshold of 25% of the original bug report's word count for the summaries, our model is unlikely to choose all the sentences found in the GSS summary for some of these summaries and this results in a low recall for the trained models.

We plan to further explore the use of the more complex features that have high variability and were not included in this study, namely clue-word score and those related to sentence entropy.

REFERENCES

- [1] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, pp. 10:1–10:35, Aug. 2011.
- [2] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, (New York, NY, USA), pp. 361–370, ACM, 2006.
- [3] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Trans. Softw. Eng.*, vol. 40, pp. 366–380, Apr. 2014.
- [4] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the 'hurried' bug report reading process to summarize bug reports," *Empirical Softw. Engg.*, vol. 20, pp. 516–548, Apr. 2015.
- [5] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "Ausum: Approach for unsupervised bug report summarization," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, (New York, NY, USA), pp. 11:1–11:11, ACM, 2012.
- [6] A. Nenkova and K. McKeown, *A Survey of Text Summarization Techniques*, pp. 43–76. Boston, MA: Springer US, 2012.
- [7] G. Murray and G. Carenini, "Summarizing spoken and written conversations," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, (Stroudsburg, PA, USA), pp. 773–782, Association for Computational Linguistics, 2008.
- [8] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: A case study of bug reports," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, (New York, NY, USA), pp. 505–514, ACM, 2010.
- [9] A. Kiani-B, M. Akbarzadeh-T., and M. H. Moeinzadeh, "Intelligent extractive text summarization using fuzzy inference systems," in *2006 IEEE International Conference on Engineering of Intelligent Systems*, pp. 1–4, April 2006.
- [10] F. Kiyomarsi, "Evaluation of automatic text summarizations based on human summaries," vol. 192, pp. 3–91, 2015. The Proceedings of 2nd Global Conference on Conference on Linguistics and Foreign Language Teaching.
- [11] W. Wang, Z. Li, J. Wang, and Z. Zheng, "How far we can go with extractive text summarization? heuristic methods to obtain near upper bounds," *Expert Systems with Applications*, vol. 90, pp. 439–463, 2017.

Generating Integration Tests Automatically Using Frequent Patterns of Method Execution Sequences

Mark Grechanik
University of Illinois at Chicago
Email: drmark@uic.edu

Gurudev Devanla
University of Illinois at Chicago
Email: gdev2@uic.edu

Abstract—Integration testing is vitally important for ensuring software quality, since many serious software defects are not isolated in single components. Unfortunately, creating integration tests is often a manual and laborious effort. A fundamental problem of software testing is how to automatically create effective integration tests with oracles that find bugs efficiently.

We created a novel approach for *Automatically Synthesizing Integration Software Tests (ASSIST)* that automatically obtains models that describe frequently interacting components in software applications, thus reducing the number of synthesized integration tests and increasing their bug-finding power. In ASSIST, static and dynamic analyses are used along with carving runtime states to obtain test input data as well as oracles for the synthesized integration tests. We experimented with three Java applications and show that integration tests that are synthesized using ASSIST have comparable bug finding power with manually created integration tests.

Index Terms—software testing, pattern mining, execution trace

I. INTRODUCTION

Many companies have adopted agile development [20], in particular continuous delivery where software is tested and released frequently, at the end of each delivery iteration. In continuous delivery, it is important to test integrated software components at the end of each short-term (i.e., ten day) iteration [21, pages 55-82]. In fact, integration testing has emerged as a major testing approach for agile and distributed software development [25], since the majority of serious software defects are no longer isolated in single components and many catastrophic problems occur in interactions among different components [23].

In *integration testing*, integrated software modules or components are evaluated as a whole to determine if they behave correctly [2, page 6] [4, page 21]. For object-oriented software, integration tests invoke methods that belong to different classes, which exchange data as a result of these invocations [11]. *Acceptance testing* is a kind of integration testing where many components of the application are integrated to implement some requirements [4]. At an extreme, *big bang testing* is an example of the coarsest granularity of the acceptance testing, where the entire application is assembled and tested in one step [33], however, its usefulness is limited, since a single debugging task is impractical [37, Section 7.2].

Executing acceptance and big-bang tests takes significant resources and time, since they require comprehensive and laborious installation and configuration of the *application under test (AUT)* on a testbed, and each run of these tests may take tens of minutes or hours, depending on the functionality of the AUT. Naturally, acceptance and big-bang tests are run on testbeds during major software releases, however, it is impractical to do so during and at the end of the short-term agile iterations. Clearly, finer granularity integration tests are required that are coarser than *unit tests*, where implementations of methods that belong to the same class are tested individually [4], [26]). Combining two or more methods that belong to different classes is an example of an integration test of a finer granularity that are needed for agile iterations [27]. Given that developers routinely make small incremental changes to applications to fix bugs and modify their functionalities, finer granularity tests are needed to verify that the AUT behaves as desired during and at the end of the agile iterations.

The larger the project, the more important integration testing is to find bugs efficiently in integrated components of an application [37]. Unfortunately, it is expensive, since the average cost per finding and fixing a defect is currently the highest for integration and acceptance testing [23]. This cost increases approximately by anywhere from five and up to twenty times if a defect is missed during integration testing and found at a later stage [15]. Creating effective integration tests requires significant time and effort, since it is not feasible to test all combinations of components in a software application, and the number of these combinations is enormous for nontrivial applications [5], [17]. Despite these difficulties, the demand for integration tests is high, since they are reported to have a higher defect removal efficiency when compared with other forms of testing (e.g., unit) [5], [23].

Our key idea is to use a stable release of the AUT to synthesize integration tests with oracles that will be used to test the subsequent releases of this AUT as part of agile iterative development. Creating integration tests be ASSISTed with automatically obtained models that describe frequently interacting components, and thus are viewed as strong candidates for containing integration bugs. ASSIST combines in a novel way static dataflow and dynamic analyses with pattern mining to guide the synthesis of integration tests. Our tool and experimental results are publicly available at <https://www.dropbox.com/s/k1mfuzfw6r7138a/assist-release.tar.gz?dl=0>.

DOI reference number: 10.18293/SEKE2019-001

II. THE PROBLEM

Making integration testing cheaper and more effective is very important and is equally very difficult. Constructing finely granular integration tests is a laborious effort that requires time and resources, which is difficult to justify especially in continuous delivery where software is released in short-term iterations. Combining different components blindly in integration tests reduces their effectiveness, since the total number of integration tests is exponential in the number of components in a software application and many components do not interact with one another. A fundamental problem of software testing is how to automatically create effective integration tests that have a high bug-finding power.

A main objective for software integration tests is to be effective in finding bugs. An equally important objective is to find bugs in a shorter time period without using significant amount of resources, i.e., software integration tests should also be efficient. A system that generates millions of random integration tests is unlikely to be effective and efficient, since running these tests will consume significant resources and time without any guarantees that integration bugs will be found. Thus, our main goal is to minimize the number of synthesized integration tests and their execution time and to increase their effectiveness of finding bugs at the same time.

III. OUR SOLUTION

The architecture and the workflow of ASSIST are shown in Figure 1. The input to ASSIST (1) is the *Application Test Suite (ATS)* that consists of the AUT, unit and acceptance tests and input test data for these tests. The AUT (2) is run using acceptance tests with the Profiler that collects Execution Traces that are (3) analyzed by the Frequent Pattern Miner that outputs (4) frequent patterns of method calls. The Model Learner (5) learns the model that correlates properties of test input data with frequently mined method calls and it produces (6) the Model that is used to prioritize the synthesis of integration tests, i.e., to produce more effective integration tests efficiently. An important contribution of ASSIST is that stakeholders concentrate on creating and improving unit and acceptance tests as they do it now, and effective integration tests will be synthesized automatically using models that are learned using these acceptance and unit tests.

To execute synthesized integration tests, input test data is required, i.e., all variables and fields of the objects should be initialized that are used in the methods of these integration tests. Moreover, since the first method of an integration test is the N^{th} method that is executed as part of some acceptance test, this method uses the values of different objects and their fields, which constitute the *state of the AUT*. Our idea is to generate test input data by carving the AUT state [39] before executing the N^{th} method in the acceptance test for the given integration test. In fact, since the sequence of methods in an integration test can be executed as part of two or more acceptance tests, the carved states for these acceptance tests will serve as the set of the input data for the same integration test. A rudimentary

state carving method is to traverse the heap starting from the objects that are created in the main method.

The Model is used (7) as the input to the Execution State Carver that reruns the AUT with specific acceptance tests in order to carve (8) AUT States for the frequent patterns of method calls. Independently from this step, (9) unit tests from the ATS are inputted to the Unit Test Analyzer that uses static analyses to obtain (10) complex Oracle Expressions that include variables and AUT classes that are eventually used in *assert* statements in these unit tests. Next, (11) specific Oracle Expressions are selected for carved AUT States and then these selected Oracle Expressions are projected (12) onto the carved AUT States using the State Projector to obtain (13) the values of the oracles. As we discussed, unit tests can contain complex expressions and control flows, and its assertion statements contain variables whose values are computed as results of executing these unit tests. To determine what objects and their fields from the AUT are used in these assertions, unit tests are analyzed using a combination of backward slicing [38] and symbolic execution to obtain all expressions that contribute to computing the values of oracle expressions in assert statements. Then, these expressions are re-evaluated given the carved states and new values for the oracles for these expressions are obtained. These obtained oracles, the source code of the AUT and the Model (14) are used by the Integration State Synthesizer that (15) outputs integration tests. This concludes the description of the ASSIST architecture.

IV. EXPERIMENTAL EVALUATION

In this section, we pose research questions (RQs), describe subject applications, explain our methodology and variables, and discuss threats to validity.

A. Research Questions

As part of evaluation, we will answer the following research questions (RQs) to assess how ASSIST meets the objectives of effectiveness and efficiency.

RQ1: How effective are synthesized integrations tests in finding bugs? The rationale for this research question is to determine whether ASSIST synthesizes integration tests that can locate more integration bugs in software when compared with certain baseline approaches, e.g., manually created integration tests and FUSION, an approach that composes integration tests from unit tests [29].

RQ2: How efficient is ASSIST in synthesizing integration tests that can help find bugs without a significant use of computing resources? The rationale for this research question is to determine if ASSIST produces fewer integration tests and their total execution time is not prohibitive, while measuring their effectiveness of finding bugs at the same time. Our goal is to show that ASSIST can synthesize a much smaller and manageable number of integration tests that have good bug-finding power.

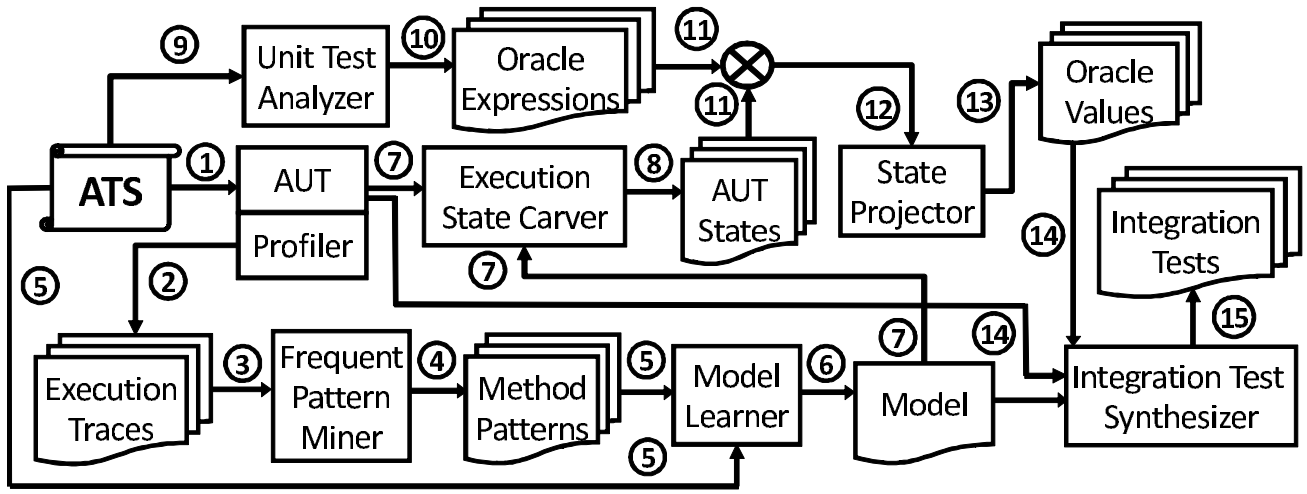


Fig. 1. The architecture and workflow of ASSIST.

TABLE I
CHARACTERISTICS OF SUBJECT APPLICATIONS.

AUT	Size, KLOC	Acc Tests	Unit Tests	Integration Tests	
				ASSIST	FUSION
NANOXML	7	36	92	48	75
SCRIBE	3.7	29	39	36	28
SHTUTXML	1.3	5	40	6	6

TABLE II
MAXIMUM VALUES OF ELAPSED EXECUTION TIME AND MEMORY
CONSUMPTION FOR ASSIST.

AUT	Instrum		Pattern Mining		Synthesis		State
	Test	Mem	Test	Mem	Test	Mem	
SCR	16s	55K	0.01s	2.3KB	18s	70K	0.2GB
NAN	17s	55K	0.04s	2.4KB	7s	60K	1.5GB
SHT	10s	57K	0.18s	2.4KB	7s	52K	0.95GB

B. Subject Applications And Their Test Suites

The subject applications for evaluating ASSIST are open-source Java applications that are widely used. Their characteristics are given in Table I. SCRIBE is one of the subject applications that serves as a OAuth module for Java applications. This application has close to 50 contributors and 22 releases. NanoXML is a Java-based non-validating parser. The third subject application is ShtutXML, an XML to text mapper. All subject applications have test suites that contain mixtures of unit and acceptance tests. In addition, we used several graduate students from UIC with prior software development experience to study these applications and create additional unit and acceptance tests.

C. Methodology And Variables

To address RQ1 and RQ2 we evaluate ASSIST using the following two dependent variables: the number of synthesized integration tests and their bug-finding power. The number of synthesized integration tests is a function of the threshold value for mining frequent method sequences. The smaller the value

is the more sequences can be mined, which is exponential in the limit. The larger the threshold value is, the fewer more frequent method sequences can be mined, however, the mining process may take a very long time. Unfortunately, frequent pattern mining algorithms are computationally intensive, and increasing the threshold value closer to one may take weeks or months even using powerful computers. Thus, as part of our experimentation we show the sizes of mined method sequences that we obtain for different threshold values and how they affect sizes of synthesized integration test suites.

We measure the bug-finding power of test suites using *mutation testing*, which is recognized as one of the strongest approaches for evaluating the effectiveness of test suites [14], [18]. The code of the *application under test* (AUT), P , is modified by applying *mutation operators* to the AUT's code to create a buggy but syntactically correct version of the AUT, P' , i.e., a *mutant*. Running the test suite for P on P' should fail some tests, i.e., the mutant is killed. Otherwise, the test suite is deemed not adequate to find bugs and it should be enhanced with new tests that can kill mutants that were not killed with the previous version of the test suite. A key measurement of the power of determining the adequacy of test suites is mutant killing ratio, i.e., the ratio of the number of mutants killed by a test suite to the total number of generated non-equivalent mutants. That is, our goal is to apply a mutation testing approach to the subject applications that generates mutants for determining the adequacy of integration test suites synthesized by ASSIST and the mutant killing ratio should be approximately the same when comparing to test suites that are manually created or generated using a competitive approach.

In our evaluation we compare the tests produced by our approach with FUSION [29], a state of the art tool for generating integration tests by combining unit tests using an object-relational model that it re-engineers from the source code of the application. The independent variables in our evaluation included the two mutation testing tools, jMINT [16] and JavaLanche [30], the set of subject applications, and the threshold values used in the pattern mining tool called BIDE

TABLE III
WE DESCRIBE SELECTED PARAMETERS FOR MINED FREQUENT
SEQUENCES FOR SYNTHESIZING INTEGRATION TESTS USING ASSIST.

AUT	Unq Mth	Max Len Seq	BIDE Inpt	Thresh	BIDE output	Seq Used In ASSIST
SCRIBE	81	60	134	0.01	129	87
NANOXML	101	602	121	0.05	180	42
SHTUTXML	28	48	298	0.002	27	15

to extract frequent method sequences from execution traces that are obtained from the subject applications. The number of integration tests that ASSIST synthesizes is also guided by the number of tests that contain oracles that come with the subject applications.

For frequent pattern mining, we use an closed sequence frequent pattern mining tool called BIDE [34]. This tool implements a very efficient algorithm for determining closed frequent sequences. We note that ASSIST is not tied to these specific tools. Any frequent pattern mining tool that identifies closed frequent sequences can be used in place of this tool.

D. Threats to Validity

One threat to external validity is the nature of applications that we used to evaluate ASSIST – they are small and may not be good representatives of the overall population of applications. We plan to conduct more experiments with many more applications to generalize these results. Subject applications were chosen based on the availability of large and diverse tests suites that we used as acceptance tests and to extract test oracles. In addition, we are also constrained by the limitations of FUSION, since it does not produce oracles. The counter argument to this threat to validity is that the subject applications are popular and have many tests and they may be viewed as good representative of the applications that are built in industry.

Another threat to validity is that we do not address the problem of storing transient state components, such as pointers to external resources (e.g., sockets and files). Since we show that our approach works for non-transient data structures, we hope to address the challenges with transient states as the next logical step in our future work. This technical challenge does not pose a threat to the core of ASSIST.

Finally, a threat to validity is that we evaluate the mutants against the integration tests generated by only one tool, namely, FUSION. Ideally, we need more tools on generating integration mutants that contain meaningful oracles, unfortunately, little research is done in this area that resulted in tools that can be used in our evaluation. The very purpose of our research is to address this limitation.

V. RESULTS

In this section, we report the results of the experiments and state how they address our RQs. We carried out experiments using Mac OS X 10.8, 64-bit CPU 2.4 GHz Intel Core i7, 8GB of RAM, 256KB L2 Cache(per core) and 6MB L3 Cache.

Performance-related measurements are shown in Table II. Clearly, the biggest space-related expense is the the carved state that requires approximately 1.5GB for NanoXML. Instrumenting the applications and synthesizing data takes less than 20 seconds, which is a reasonable expense. We can conclude that with respect to time and memory consumption, ASSIST is a practical approach that can be used for reasonably sized software applications to generate integration tests.

We evaluate the results of our approach and compare its effectiveness compared to manually generated tests and the tests generated by FUSION. Results of our experiments are shown in Table IV. For the AUT SCRIBE the strong mutant killing ratio is the same that is achieved with manually created tests. This result shows that it is possible to achieve the same result with ASSIST as it is achieved with manually created tests. Even though the ratio for SHTUTXML is smaller with ASSIST-based integration tests when compared to manually created tests, we view it as an impressive result, since ASSIST is fully automatic.

The application NANOXML is a source of concern, since its mutant killing ratio is very low with ASSIST-based integration tests. Our investigation revealed the following. When we run the acceptance tests, that are 101 unique methods in NANOXML that are invoked and are written in execution traces. Of these, there are 27 unique methods that are mined as frequent sequential patterns. Unfortunately, there are unit tests only for five methods. For the remaining methods there are no assertions in any of the manually created tests. Moreover, those methods that are used in unit tests and are part of the mined frequent sequences are loosely integrated with other methods in these sequences, so that the tightness of integration is very poor. ASSIST is based on the assumption, that unit tests and acceptance tests are adequate to produce integration tests that can be used as the software evolves. This also exposes the characteristics of the test suite and the lack of adequate tests for methods that were identified in the set of frequently used methods. As a result, ASSIST could not synthesize enough effective integration tests.

The case with NANOXML highlights the results of consequences of deficiencies that result in poorly built tests rather than limitations of ASSIST. Indeed, if acceptance and unit tests are abundant and if they are constructed properly, ASSIST automates a difficult and laborious tasks of creating effective integration tests. Moreover, for SCRIBE, ASSIST achieve the same mutant killing ratio with fewer synthesized tests, i.e., 36 when compared with 100 manually created tests. For SHTUTXML, the mutant killing ratio is achieved with only six synthesized tests.

We also see that using weak mutation ASSIST produces a weak mutation killing ratio that is equal to the manually created tests for two applications, SCRIBE and NANOXML. In addition, our approach performs equally well compared to FUSION for both these applications. In both these cases, our approach also provides mutant killing ratios that is at least 50% of mutant killing ratio of manually created tests.

The results of evaluating ASSIST with the mutation tool

TABLE IV
EVALUATING THE EFFECTIVENESS OF ASSIST BY COMPARING IT WITH FUSION. MUTANTS ARE GENERATED USING JMINT.

AUT Name	Type of tests	Tests Created	Mutants Generated	Mutants Triggered	Mutants Killed	Trigger Ratio	Killed Ratio
SCRIBE	DEFAULT	100	115	15	6	13.04	5.21
	ASSIST	36	115	15	6	13.04	5.21
	FUSION	28	115	15	-	13.04	
NANOXML	DEFAULT	116	104	95	34	91.34	32.69
	ASSIST	48	104	5	2	4.80	1.92
	FUSION	75	104	40	-	38.46	
SHTUTXML	DEFAULT	43	12	7	7	58.3	58.3
	ASSIST	6	12	2	2	16.66	16.66
	FUSION	6	12	2	-	16.66	

TABLE V
EVALUATING THE EFFECTIVENESS OF ASSIST BY COMPARING IT WITH FUSION. MUTANTS ARE GENERATED USING JAVALANCHE.

Description of the Step	SCRIBE				NANOXML				SHTUTXML			
	FUSION		ASSIST		FUSION		ASSIST		FUSION		ASSIST	
Total mutations	356		2634		1442		7330		193		719	
Covered Mutations in Scan Step	105	29.49%	2111	80.14%	731	50.69%	6957	94.91%	0	0	403	56.05%
Covered Mutations	100	28.09%	100	3.80%	99	6.87%	99	1.35%	0	0	100	13.91%
Not Covered Mutations	256	71.91%	2534	96.20%	1343	93.13%	7231	98.65%	193	0	619	86.09%
Killed Mutants	36	10.11%	80	3.04%	47	3.26%	66	0.90%	0	0	49	6.82%
Surviving Mutants	320	89.89%	2554	96.96%	1395	96.74%	7264	99.10%	193	0	670	93.18%
Mutations Score	10.11%		3.04%		3.26%		0.90%		0.0%		6.82%	
Mutation Score for covered mutants	36.00%		80%		47.47%		66.67%		0		49%	

called Javalanche are shown in Table V. ASSIST-based synthesized integration tests are richer than ones created by FUSION, resulting in more mutants generated by Javalanche for ASSIST. As a result, ASSIST-based tests cover more mutated statements when compared to ones generated by FUSION. ASSIST-based tests kill many more mutants when compared with FUSION-generated tests. The mutation score for covered mutants is much higher for all applications for ASSIST. Based on this evidence we can conclude that **we positively answer RQ1, i.e., synthesized integrations tests are effective in finding bugs.**

To address RQ2, we will need to evaluate the results of our approach in terms of performance of the frequent mining algorithm which is dependent on the initial sequences generated during the execution of acceptance tests and the number of tests ASSIST generates. Table III provides information on the number of unique methods for each subject application, maximal sequence of method calls that was traced during execution of acceptance tests. The last three columns state the threshold values that were used to identify the frequent sequences, and the actual number of sequences that were used by ASSIST. For each of the subject applications, we note that it did not take more than a couple of seconds to identify these frequent sequences. The other result we need to evaluate to address RQ2 would be the number of integration tests ASSIST produces. Table I provides details on the number of tests we produce. It can be observed that the number of tests generated by ASSIST is close to the number of tests that were created manually or by FUSION. This shows that even though there is a potential of execution trace producing a large number of sequences, using a combination frequent mining and static analysis techniques ASSIST is able to produce tests that can

be executed as efficiently as manually produced tests and the tests produced by FUSION. Based on this evidence we can conclude that **we positively answer RQ2, i.e., ASSIST is efficient in synthesizing integration tests that can help find bugs without a significant use of computing resources.**

VI. RELATED WORK

Related work on automatic generation of integration tests has many branches. Different model-based approaches exist for generating integration tests using different types of formal models [9], [19], [28] and using modeling approaches for creating integration tests for distributed applications [6], [35]. Some approaches learn models and class dependencies for integration testing from application code and behavior [3] or from previous versions of the same application [7], [8]. Opposite to these approaches, ASSIST does not require models for generating integration tests, since models may be outdated, incomplete, or they may not exist at all. In addition, it is not clear how oracles are defined for these approaches. ASSIST is different from these approaches and complementary to them in that ASSIST synthesizes integration tests using models of method invocations. Unlike ASSIST, these approaches do not have explicit mechanisms to control the number of generated tests while increasing their effectiveness, and it is a question how efficiently it is achieved.

Class Integration and Test Order (CITO) approaches determine orders in which classes are composed in integration tests using graph-based and search-based solutions [1], [10], [12], [13], [22], [36]. However, the determination of a cost function, which is able to generate the best solutions, is not always a trivial task. In contrast, ASSIST does not require cost functions, and it extracts oracles from tests, while CITO

does not address the oracle problem. Unlike ASSIST, CITO approaches do not reflect the frequency of executions of integrated components, and therefore it is not clear how it can increase the effectiveness of the composed integration tests.

Approaches that compose integration tests from unit tests and some software modules [24], [29], [31], [32] address a different angle of the problem than ASSIST – the latter concentrate on producing effective integration tests efficiently, while it is unclear how the former approach addresses a problem of reducing the large number of combinations of unit tests into integration tests.

VII. CONCLUSION

We created a novel approach for *Automatically Synthesizing Integration Software Tests (ASSIST)* that automatically obtains models that describe frequently interacting components in software applications, thus reducing the number of synthesized integration tests and increasing their bug-finding power. In ASSIST, static and dynamic analyses are used along with carving runtime states to obtain test input data as well as oracles for the synthesized integration tests. We experimented with three Java applications and show that integration tests that are synthesized using ASSIST have comparable bug finding power with manually created integration tests.

REFERENCES

- [1] A. Abdurazik and J. Offutt. Coupling-based class integration and test order. *AST '06*, pages 50–56, New York, NY, USA, 2006. ACM.
- [2] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 2008.
- [3] L. Badri, M. Badri, and V. S. Ble. A method level based approach for oo integration testing: An experimental study. *SNPD-SAWN '05*, pages 102–109, Washington, DC, USA, 2005. IEEE Computer Society.
- [4] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, 2nd edition, 1990.
- [5] A. Bertolino, P. Inverardi, H. Muccini, and A. Rosetti. An approach to integration testing based on architectural descriptions. *ICECCS '97*, pages 77–, Washington, DC, USA, 1997. IEEE Computer Society.
- [6] A. Bertolino and A. Polini. Soa test governance: Enabling service integration testing across organization and technology borders. *ICSTW '09*, pages 277–286, Washington, DC, USA, 2009. IEEE Computer Society.
- [7] L. Borner and B. Paech. Using dependency information to select the test focus in the integration testing process. *TAIC-PART '09*, pages 135–143, Washington, DC, USA, 2009. IEEE Computer Society.
- [8] L. Borner and B. Paech. Using dependency information to select the test focus in the integration testing process. *TAIC-PART '09*, pages 135–143, Washington, DC, USA, 2009. IEEE Computer Society.
- [9] L. Briand, Y. Labiche, and Y. Liu. Combining uml sequence and state machine diagrams for data-flow based integration testing. *ECMFA '12*, pages 74–89, Berlin, Heidelberg, 2012. Springer-Verlag.
- [10] L. C. Briand, Y. Labiche, and Y. Wang. An investigation of graph-based class integration test order strategies. *IEEE Trans. Softw. Eng.*, 29(7):594–607, July 2003.
- [11] P. J. Clarke. *A taxonomy of classes to support integration testing and the mapping of implementation-based testing techniques to classes*. PhD thesis, Clemson, SC, USA, 2003. AAI3098273.
- [12] R. Da Veiga Cabral, A. Pozo, and S. R. Vergilio. A pareto ant colony algorithm applied to the class integration and test order problem. *ICTSS'10*, pages 16–29, Berlin, Heidelberg, 2010. Springer-Verlag.
- [13] R. Delamare and N. A. Kraft. A genetic algorithm for computing class integration test orders for aspect-oriented systems. *ICST '12*, pages 804–813, Washington, DC, USA, 2012. IEEE Computer Society.
- [14] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, Apr. 1978.
- [15] Gartner. It key metrics data 2008: Key applications measures: Current year: Defect rates. http://www.gartner.com/DisplayDocument?ref=g_search&id=557525, Dec. 2007.
- [16] M. Grechanik and G. Devanla. Mutation integration testing. In *2016 IEEE International Conference on Software Quality, Reliability and Security, QRS 2016, Vienna, Austria, August 1-3, 2016*, pages 353–364, 2016.
- [17] M. Greiler, A. v. Deursen, and M.-A. Storey. Test confessions: a study of testing practices for plug-in systems. *ICSE 2012*, pages 244–254, Piscataway, NJ, USA, 2012. IEEE Press.
- [18] R. G. Hamlet. Testing programs with the aid of a compiler. *IEEE Trans. Softw. Eng.*, 3(4):279–290, July 1977.
- [19] J. Hartmann, C. Imoberdorf, and M. Meisinger. Uml-based integration testing. *ISSTA '00*, pages 60–70, New York, NY, USA, 2000. ACM.
- [20] J. Highsmith and A. Cockburn. Agile software development: The business of innovation. *IEEE Computer*, 34(9):120–122, 2001.
- [21] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 1st edition, 2010.
- [22] Z. Jin and A. J. Offutt. Coupling-based integration testing. *ICECCS '96*, pages 10–, Washington, DC, USA, 1996. IEEE Computer Society.
- [23] C. Jones and O. Bonsignour. *The Economics of Software Quality*. Addison-Wesley Professional, Aug. 2011.
- [24] M. Jorde, S. Elbaum, and M. B. Dwyer. Increasing test granularity by aggregating unit tests. *ASE '08*, pages 9–18, Washington, DC, USA, 2008. IEEE Computer Society.
- [25] M. Luke. How early integration testing enables agile development. <http://www.ibm.com/developerworks/rational/library/early-integration-testing-enables-agile-development>, June 2012.
- [26] A. P. Mathur. *Foundations of Software Testing*. Addison-Wesley Professional, 1st edition, 2008.
- [27] A. J. Offutt. Unit testing versus integration testing. pages 1108–1109, Washington, DC, USA, 1991. IEEE Computer Society.
- [28] S. Ogata and S. Matsuura. A method of automatic integration test case generation from uml-based scenario. *WSEAS Trans. Info. Sci. and App.*, 7(4):598–607, Apr. 2010.
- [29] M. Pezzè, K. Rubinov, and J. Wuttke. Generating effective integration test cases from unit ones. In *Proc. of 6th IEEE ICST*, 2013.
- [30] D. Schuler and A. Zeller. Javalanche: efficient mutation testing for java. *ESEC/FSE '09*, pages 297–298, New York, NY, USA, 2009. ACM.
- [31] S.-H. Shin, S.-K. Park, K.-H. Choi, and K.-H. Jung. Normalized adaptive random test for integration tests. *COMPSACW '10*, pages 335–340, Washington, DC, USA, 2010. IEEE Computer Society.
- [32] Y. Shin, Y. Choi, and W. J. Lee. Integration testing through reusing representative unit test cases for high-confidence medical software. *Comput. Biol. Med.*, 43(5):434–443, June 2013.
- [33] J. A. Solheim and J. H. Rowland. An empirical study of testing and integration strategies using artificial software systems. *IEEE Trans. Softw. Eng.*, 19(10):941–949, Oct. 1993.
- [34] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. *ICDE '04*, pages 79–, Washington, DC, USA, 2004. IEEE Computer Society.
- [35] S. Wang, Y. Ji, W. Dong, and S. Yang. A new formal test method for networked software integration testing. *ICCSA'10*, pages 463–474, Berlin, Heidelberg, 2010. Springer-Verlag.
- [36] Z. Wang, B. Li, L. Wang, and Q. Li. An effective approach for automatic generation of class integration test order. *COMPSAC '11*, pages 680–681, Washington, DC, USA, 2011. IEEE Computer Society.
- [37] A. H. Watson and T. J. McCabe. Structured testing: A testing methodology using the cyclomatic complexity metric. NIST Special Publication 500-235, NIST: National Institute of Standards and Technology, Gaithersburg, MD, 1996. See <http://hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/235/title.htm>.
- [38] M. Weiser. Program slicing. *ICSE '81*, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.
- [39] G. Xu, A. Rountev, Y. Tang, and F. Qin. Efficient checkpointing of java software using context-sensitive capture and replay. *ESEC-FSE '07*, pages 85–94, New York, NY, USA, 2007. ACM.

CrashAwareDev: Supporting Software Development based on Crash Report Mining and Analysis

Leandro Beserra^{*†}, Roberta Coelho[†]

^{*}Informatics Superintendence

[†]Department of Informatics and Applied Mathematics

Federal University of Rio Grande do Norte

Natal, Brazil

ldbessera@info.ufrn.br, roberta@dimap.ufrn.br

Abstract—Exception handling mechanisms are a common feature of mainstream programming languages to deal with exceptional conditions. If on the one hand they allow the programmer to prepare the code to deal with exceptional conditions, on the other hand they can become a source of bugs that threatens the application robustness – studies have shown that uncaught exceptions are the main cause of application crashes. To keep track of crashes and enable an easier fault localization, several applications, nowadays, use crash reporting tools. In this work, we propose a tool named CrashAwareDev, integrated with the Eclipse IDE, which mines the information available in crash reporting tools to support programmers in their day-to-day activities. The proposed tool alerts developers about the classes related to recent crashes and warns about source code characteristics (bug patterns) that can be related to future crashes (similar to the ones that have happened). Doing so the tool aims at bringing the development environment closer to crash reporting tools, that are usually only used for bug fixing or for extracting robustness metrics. A case study was conducted and showed that the tool can support software development by displaying bug pattern alerts directly in the source code, signaling the classes involved in recent faults, and speeding up the crash’s fault localization within the development environment itself.

Index Terms—Exception handling, crash report, Eclipse plug-in, uncaught exceptions

I. INTRODUCTION

Exception handling mechanisms [1] are a common feature of modern programming languages. Exception handling structures are used to deal with unexpected events that occur during the execution

of a program [2], allowing exceptions to be thrown, captured, and handled at different points in the system. However, the exception handling code designed to make a system more robust often works the other way around and become a burden programmers has to cope with, leading to bugs such as the uncaught exceptions. The uncaught exceptions are the main cause of crashes in Java software systems [3]. A crash is an abnormal behavior of a system that leads to the interruption of its execution.

After a crash occurs, systems typically store information related to the crash on crash report systems. Such information usually contains the uncaught exception that caused the crash and its stack trace. The exception stack trace is a representation of the method call stack and contains information about classes and methods by which an exception was propagated. Since the exception stack trace is a source of information widely used by programmers while debugging they are often added on crash reports [4]. Moreover, these reports may contain other information such as the operating system used at the time of the crash, the user login, the browser/system version, the user’s IP address, and other request parameters. In addition to facilitating the fault localization, the information available on crash report tools can assist in prioritizing bug corrections (depending on the number of users affected, for example) or understanding the impact of system crashes [5]. The utility of crash report systems may go beyond recording crash data and supporting debugging. Some studies have shown

that such information can be used for various purposes such as fault classification [6][7] and fault localization [8][9][10]. None of the existing works, however, extract data from crash reports to support programmers during system coding.

In this paper, we propose a way to mine information stored in crash reports and provide useful information to the programmer within his/her programming environment. We present a tool called CrashAwareDev, an Eclipse¹ IDE plug-in, which supports the developer in coding time by (i) alerting him/her about the classes related to recent crashes; (ii) warning about source code characteristics (bug patterns) that can be related to future crashes (similar to the ones that have happened); and (iii) providing direct access to crash reports within the IDE. A case study was conducted on an industrial web-based software system comprising of 1300 KLOC of Java source code. Along the evaluation period, a group of 5 developers used the tool on a daily basis (during 4 days). Overall the tool presented 95 warnings (i.e., 17 class alerts and 78 bug pattern warnings).

II. BACKGROUND

The Java programming language provides an exception handling mechanism to support error handling [12]. When an error occurs during execution of code in a try block, the error is caught and handled by an exception handler in one of the subsequent catch blocks associated with it. If no catch block can handle the error, the method is terminated abnormally and the Java virtual machine (JVM) searches backward through the call stack to find an exception handler that can handle the error [15].

In Java, exceptions are represented according to a class hierarchy, on which every exception is an instance of the Throwable class, and can be of three kinds: the checked exceptions (extends Exception), the runtime exceptions (extends RuntimeException) and errors (extends Error) [12].

Checked exceptions represent conditions that, although exceptional, can reasonably be expected to occur, and if they do occur must be dealt with in some way. Unchecked runtime exceptions

represent conditions that, generally speaking, reflect errors in your program's logic and cannot be reasonably recovered from at run time [13]. By convention, instances of Error represent unrecoverable conditions which usually result from failures detected by the Java Virtual Machine due to resource limitations, such as OutOfMemoryError. Normally these cannot be handled inside the application [13].

III. ANALYZING CRASH REPORTS

A. Methodology

Before implementing the CrashAwareDev tool, we conducted a study whose goal was to identify the characteristics of most frequent crashes of an industrial Web-based system, and based on such characteristics, check whether or not existing static analysis tools could alert about them. Figure 1 illustrates the main steps taken in this study. The target system used in this study was an industrial Web-based system, named SIPAC, comprising of 1300 KLOC of Java source code and designed to automate business processes for universities focusing on different and complementary aspects, such as administration, planning and management – SIPAC is used in approximately 55 Brazilian universities.

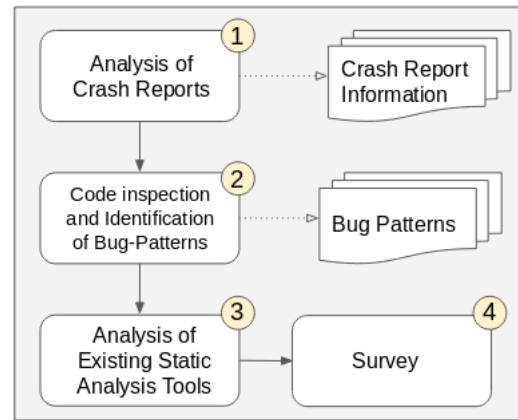


Fig. 1. Methodology overview.

Step 1: Analysis of Crash Reports. We analyzed the most frequent exceptions that caused crashes over a period of one month.

Step 2: Code Inspection and Identification of Bug Patterns. We manually inspected the code related to the most frequent kinds system crashes to identify common characteristics among them.

¹<https://www.eclipse.org/>

Step 3: Analysis of Existing Static Analysis Tools. We analyzed whether or not existing static analysis tools could be able to detect and therefore prevent such causes of crashes identified on the previous step.

Step 4: Survey. Finally, we conducted a study on which we surveyed a group of developers (working on the target system) to evaluate how they used.

B. Crash Analysis

In order to investigate the main causes of crashes reported for the target system, we analyzed the crash reports of one month (June 2018). In this period (2018-06-01 to 2018-06-30), approximately 1933617 accesses were made to the system, from this set 680 accesses were faulty (0.0035% of the accesses) - on which the user faced one or more crashes. We observed that approximately 1966 crashes were recorded in the period, affecting 347 distinct users.

We extracted the types of most frequent uncaught exceptions (leading to crashes). The top 5 types of uncaught exceptions raised shown in Table I. We can observe that approximately 63.9% (1241 errors of 1966) of the crashes of the analyzed period was caused by five types of exceptions.

Root Cause	Ocurr.	%	Analyzed
NullPointerException	749	38,09	10
LazyInitializationException	171	8,69	2
JspException	166	8,44	21
IllegalArgumentException	102	5,18	1
SQLException	53	2,69	5
Total	1241	63,9	39

TABLE I

THE TOP 5 TYPES OF UNCAUGHT EXCEPTIONS RAISED IN JUNE/2018.

We then manually inspected the source code related to a subset of such crashes (we randomly selected a subset of each of the top 5 uncaught exceptions – as shown in the last column of Table I). The purpose of this analysis was to identify common characteristics related to the causes of the most frequent uncaught exceptions.

Overall we inspected the source code related to 39 crashes. We could observe common characteristics among some of them, which led to the

definition of four specific bug patterns listed in Table II.

Tag	Description	Ocurr.
BP01	Unchecked database query return	3
BP02	Unchecked classes methods parameters	2
BP03	Unchecked request parameters	2
BP04	Controllers do not implement get/set methods	4

TABLE II

BUG PATTERNS DETECTED DURING ANALYSIS.

Each bug pattern found in this study is described as follows:

- **BP01:** When querying any data in a database, it is recommended to check if the value is not null, to prevent a possible null pointer exception.
- **BP02:** Static methods of utility classes should check if their arguments are not null.
- **BP03:** Like BP01, objects retrieved from the HTTP session must be checked.
- **BP04:** Private attributes of view controllers must most often have the get and set methods implemented. Otherwise, a JspException may be thrown while rendering the Web page.

C. Analysis of Existing Static Analysis Tools

We then investigated whether some of the existing static analysis tools could alert about some of the 39 crash causes identified during the manual inspection. We used SonarLint² and SpotBugs³, both in its Eclipse's plug-in version and observed that none of the 39 defects were detected by the tools.

D. Survey

We applied a survey the developers working on the target system to investigate how they use the crash report information during development. The survey was sent to 25 developers, and we obtained 14 responses, from which: 5 respondents mentioned that never used the crash report; 8 respondents mentioned that only used for debugging purposes; and only one mentioned that used the crash report on a daily basis for monitoring of errors. Moreover,

²<https://www.sonarlint.org/>

³<https://spotbugs.github.io>

7 respondents mentioned that they had difficulties in using the features of the crash report system used.

IV. THE CRASHAWAREDEV TOOL

Based on the steps described previously we implemented a tool – called CrashAwareDev. CrashAwareDev’s main purpose is to present information mined from crash reports and identified bug patterns on the developer’s IDE, at coding time. One of the motivations of this research was the perception that, in the context of target system development (SIPAC), crash report information is basically used for debugging failures and was not used to alert the developer of potential crash causes or classes frequently related to crashes. Figure 2 represents an overview of the main features of CrashAwareDev and described next.

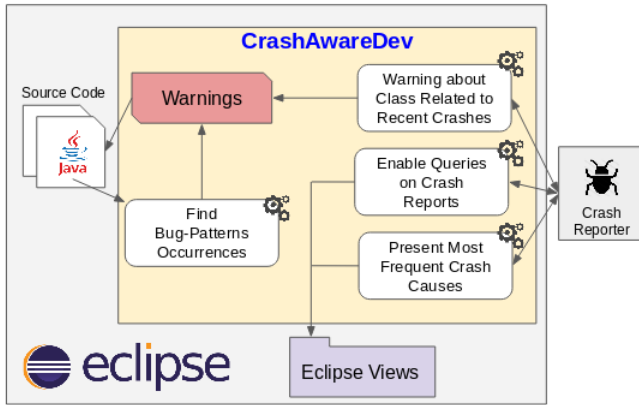


Fig. 2. CrashAwareDev’s Components diagram.

Warning about Classes related to Recent Crashes. The tool alerts whether the class being changed was associated with at least one recent system crash (i.e., the period considered is configurable). We have adopted the following heuristic: if the class appears in the exception stack trace of one crash, then it will be associated with the crash, as this means that the exception at some point was propagated within a method of this class. However, this is not to say that the defect is located in that class, but can information help to identify the fault’s origin. The tool generates warnings that are displayed in source code during development as illustrated in Figure 3.

Find Bug-Patterns Occurrences. On the study described previously, we identified source code

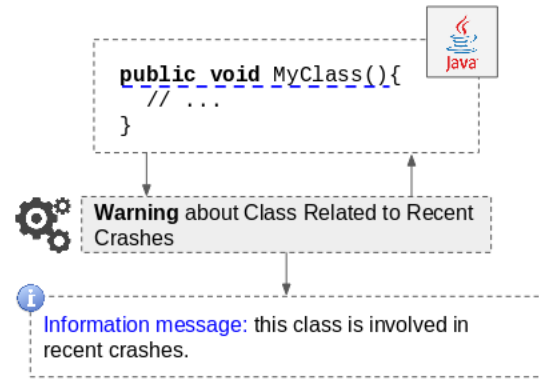


Fig. 3. Warning about classes related to recent crashes.

characteristics (bug patterns) that lead to crashes in SIPAC. Hence, at each compilation, the tool statically analyzes of the changed artifacts looking for bug patterns (described in Table II) and warnings that are displayed directly in IDE as illustrated in Figure 4.

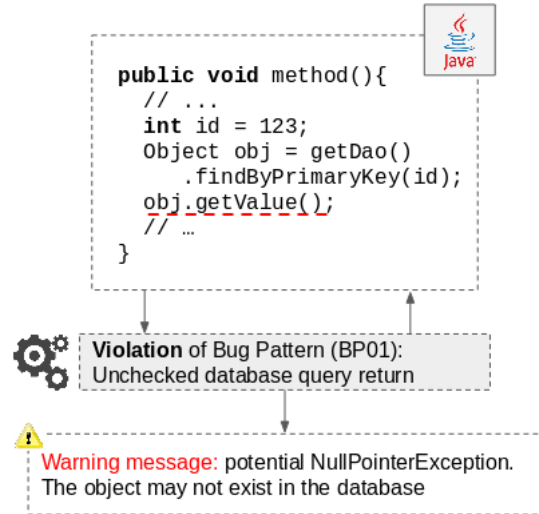


Fig. 4. Bug-Pattern Violation Alert.

Enable Queries on Crash Reports. This feature aims to bring the programmer closer to the crash report during coding. The goal is to enable the developer to query for recent system crashes, filtering them by class name. Summarized crash information is displayed in an Eclipse view with a link to display the complete information directly in the crash reporter (in an external browser).

Present Most Frequent Crash Causes. This feature basically displays the most common types of uncaught exceptions that have occurred in a

given period of time (e.g., one month). The data displayed are similar to those shown in Table I.

V. EVALUATION

To evaluate CrashAwareDev tool we performed a case study on which a group of developers used the tool for a given period of time and quantitative and qualitative data regarding the tool usage was collected. We invited SIPAC developers to participate in the study and a group of 5 developers volunteered to participate. They received brief training on the features of CrashAwareDev, and used the tool for 4 consecutive days, for approximately 8 hours daily. Table III presents the metrics collected in this study.

Metric	Count
M1: Number of classes changed	105
M2: Number of classes with some alert	61
M3: Number of alerts per bug pattern	78
BP01 - Check values when querying in a database	47
BP02 - Check auxiliary methods arguments	15
BP03 - Check values carried by request	16
BP04 - Implement private attributes' get/set	N/A
M4: Class alerts associated with a crash	17

TABLE III
METRICS COLLECTED DURING EVALUATION

During the execution period, we collected in the logs that 105 different classes were changed (M1). In 61 of these, some problem was detected by the tool and at least one warning was shown to the developer (M2). A total of 78 bug pattern alerts were displayed. They were distributed among the patterns listed in Table III (M3).

The BP04 pattern was removed during the execution of the study because the participants reported that most warnings would be false positives, as not every private method of controllers should necessarily be referenced in JSP pages. Therefore, we disabled the checking of this pattern during the study. As described earlier, the *Class Checker* feature analyzes whether the changed class is present in some recent crash stack trace. We collected 17 warnings on this check (M4).

Moreover, we also counted how many times each CrashAwareDev feature was used during the study (Q5). The *Query Crashes from Crash Reporter* function was used four times by the participants. We observed that this query was performed

after an alert was displayed in the class (during the *Class Checker* execution) and the developer was interested in checking the crashes in which the class was involved. Of these four queries, in two times the participant used the link to see the complete information of the crash. The *Query Top Root Causes* feature was not used during this study. This information did not prove useful to programmers during the study, but we believe that it is of greater interest to leader developers (who did not participate in the study).

We also questioned participants about the usefulness of the tool. All of them mentioned that the tool warnings were useful. One of the participants also mentioned that more bug-checking rules could have been implemented as they could indeed help in preventing crashes in the long run. Two other developers also suggested that in some cases the tool might also support the fault localization in the production environment in a future version.

VI. RELATED WORK

Information mining on crash reporters and/or stack traces. Some studies were carried out with the objective of extracting data from crash reports and using them for various purposes. Among them are those who used the data to find bug hazards in exception handling code [14], to classify the types of failures [6][7] and to facilitate their location [8][9][10]. In this article, we use a grouping of crashes per type of exception. We sought to identify error patterns associated with these types through manual analysis of crashes. Our research did not propose any mechanism for locating defects, but we tried to approach the programmer's development environment to the crash reporter, which may help at coding time the identification of classes associated to recent faults.

Bug Detection Tools. Several static analysis tools have been proposed to detect bug patterns in the source code from predefined rules. These patterns are categorized into various types such as correctness, code smells, vulnerability, security, performance, etc. PMD [17] is an open-source application and is based on bug patterns to find errors in the Java source code. It allows new patterns to be written in Java or XPath. Another tool proposed was the FindBugs [18] which also

examines the source code and bytecode of Java programs. In this paper a subset of the FindBugs rules was described and compared to the PMD with respect to the number of alerts generated, showing that the number of FindBugs alerts is lower in all experiments done. In 2016 FindBugs was discontinued and succeeded by SpotBugs, which we use in this work. Another existing static analysis tool is SonarQube [19] which presented a proposal for analysis along with continuous integration systems. We used in this work your Eclipse version, called SonarLint. CrashAwareDev has a static analysis feature, as well as the tools mentioned. However, we seek to define rules based on real crashes, in order to reduce the known false positives in the existing tools.

VII. CONCLUDING REMARKS

In this work, we presented a way to use data from crash reports to support programmers during software development. To promote this support, we have developed a plug-in tool for the Eclipse IDE, CrashAwareDev. Before proposing the tool, we performed a detailed study of crashes stored in a real crash reporter in order to identify the main causes of crashes. A set of common causes of crashes were defined as bug patterns which could be identified in the static analysis performed by CrashAwareDev. Moreover, the tool also alerts the developer about classes frequently involved in crashes and enabled them to access information of crash reports within the IDE. A case study in a real development context was performed and the results revealed that the tool could indeed alert the developers about several application-specific bug patterns.

Acknowledgments. This research has been supported by CAPES-BRAZIL.

REFERENCES

- [1] Goodenough J. B. Exception handling: Issues and a proposed notation. *Commun.ACM*, ACM, New York, NY, USA, v. 18, n. 12, p. 683–696, dez. 1975. ISSN 0001-0782.
- [2] Sawadpong P.; Allen E. B. Software defect prediction using exception handling call graphs: A case study. In: 2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE). [S.l.: s.n.], 2016. p. 55–62. ISSN 1530-2059.
- [3] Jo J.-W. et al. An uncaught exception analysis for java. *Journal of Systems and Software*, v. 72, n. 1, p. 59 – 69, 2004. ISSN 0164-1212.
- [4] Schroter A. et al. Do stack traces help developers fix bugs? In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). [S.l.: s.n.], 2010. p.118–121. ISSN 2160-1852.
- [5] An L.; Khomh F. Challenges and issues of mining crash reports. In: 2015 IEEE 1st International Workshop on Software Analytics (SWAN). [S.l.: s.n.], 2015. p. 5–8.
- [6] Kim S.; Zimmermann T.; Nagappan N. Crash graphs: An aggregated view of multiple crashes to improve crash triage. In: Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks. Washington, DC, USA: IEEE Computer Society, 2011. (DSN '11), p. 486–493. ISBN 978-1-4244-9232-9.
- [7] Dhaliwal T.; Khomh F.; Zou Y. Classifying field crash reports for fixing bugs: A case study of mozilla firefox. In: Proceedings of the 2011 27th IEEE International Conference on Software Maintenance. Washington, DC, USA: IEEE Computer Society, 2011. (ICSM '11), p. 333–342. ISBN 978-1-4577-0663-9
- [8] Sinha S. et al. Fault localization and repair for java runtime exceptions. In: Proceedings of the Eighteenth International Symposium on Software Testing and Analysis. New York, NY, USA: ACM, 2009. (ISSTA '09), p. 153–164. ISBN 978-1-60558-338-9.
- [9] Wang S.; Khomh F.; Zou Y. Improving bug localization using correlations in crashreports. In: 2013 10th Working Conference on Mining Software Repositories (MSR). [S.l.:s.n.], 2013. p. 247–256. ISSN 2160-1852.
- [10] Wu R. et al. Crashlocator: Locating crashing faults based on crash stacks. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis. New York, NY, USA: ACM, 2014. (ISSTA 2014), p. 204–214. ISBN 978-1-4503-2645-2.
- [11] Cabral B., Marques P. (2007) Exception Handling: A Field Study in Java and .NET. In: Ernst E. (eds) ECOOP 2007 – Object-Oriented Programming. ECOOP 2007. Lecture Notes in Computer Science, vol 4609. Springer, Berlin, Heidelberg
- [12] Gosling J, Joy B, Steele G. The Java Language Specification (The Java Series). Addison-Wesley: Reading, MA, 1997.
- [13] Arnold K., Gosling J., Holmes D., The Java Programming Language, Fourth Edition, Addison-Wesley Professional, 2005.
- [14] Coelho R. et al. Unveiling exception handling bug hazards in android based on github and google code issues. In: Proceedings of the 12th Working Conference on Mining Software Repositories. Piscataway, NJ, USA: IEEE Press, 2015. (MSR '15), p. 134–145.
- [15] Lee, S. , Yang, B. and Moon, S. (2004), Efficient Java exception handling in just-in-time compilation. *Softw: Pract. Exper.*, 34: 1463-1480. doi:10.1002/spe.622
- [16] Basili V. R.; Caldiera G.; Rombach D. The goal question metric approach. *Encyclopedia of Software Engineering*, v. 1, 01 1994.
- [17] PMD: An extensible cross-language static code analyzer. 2018. <https://pmd.github.io/> [June 2018].
- [18] Hovemeyer D.; Pugh W. Finding bugs is easy. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 39, n. 12, p. 92–106, dez. 2004. ISSN 0362-1340.
- [19] Sonarsource. SonarLint. 2019. <https://www.sonarlint.org/> [January 2019].

Dynamic and Interoperable Control of IoT Devices and Applications based on Calvin Framework

Fernanda Famá, Cleuves Cajé de Carvalho, Danilo F. S. Santos, Angelo Perkusich, Kyller Gorgônio

Embedded Lab, Federal University of Campina Grande, Campina Grande, Brazil
{cleuves.carvalho, fernanda.fama, danilo.santos, perkusic, kyller}@embedded.ufcg.edu.br

Abstract

The development of tools and smart devices has grown exponentially over the past few years, most due to the appearance of the Internet of Things (IoT). The large number of different connected devices highlighted challenges such as interoperability, scalability and reliability. In this scenario, for the development of new services and applications, it is necessary to use software platforms that ease the deployment and validation based on those challenges. With this, arises the need for middlewares, platforms that provide an environment for the development of such applications. In this way, in order to provide a simple, lightweight and dynamic approach, this article presents a study and development of an architecture that allows the communication, control and monitoring of IoT devices using an intuitive manner through an actor model. This is possible through the integration of the Calvin framework, the MQTT protocol and the OCF data model, providing an interoperable, reliable, dynamic and remote communication. A smart home environment was used for validation showing the relevance of the proposal.

Actor model; dynamic control; interoperability; IoT; MQTT; OCF; smart home

1 Introduction

Internet of Things (IoT) is described as a network infrastructure with interoperable communication standards and protocols where physical or virtual “things” share information in real time [22]. Aiming to make the Internet more comprehensive and immersive, IoT enables access to a wide range of devices. Those devices are used to build applications in many domains, including industrial and home automation, intelligent cities and healthcare [24].

Due to the increasing interest in IoT application, a wide variety of devices and objects that aims to provide utilities and services through the Internet are under development. This variety of devices and services creates new challenges, which demands the use of different architecture models in terms of Software Engineering.

The main challenges to achieve the maximum potential of IoT are interoperability, mobility, scalability, performance, security and privacy. Because of the heterogeneity of devices and platforms, interoperability is a major obstacle to be overcome. Standardization of protocols and pattern interpretations are important to make all devices accessible and interoperable. Therefore, data exchange between a large number of devices must be managed so that sensitive data is not compromised [1, 10].

Middlewares are used to allow the usage of heterogeneous components and to abstract implementation details of network protocols and communication resources. A middleware works as a communication link between devices and applications for IoT. It should provide interoperability, device discovery and management, security, privacy, and also high-volume data management [2, 15].

In this paper, we propose a new IoT System architecture model, based on an actor-based middleware, that is interoperable, dynamically controllable and manageable. This new system is built on top of Calvin [21] middleware, MQTT communication protocol, and oneIoTa models based on the OpenConnectivity Foundation (OCF)¹ specifications. While Calvin was used for the development, deployment and execution of an IoT smart home application, MQTT and oneIoTa provided interoperability and standardization between different devices. As a validation scenario, we built a smart-home application, which, based on the user behaviour, demands ways to dynamic control remote devices in a interoperable way.

The paper is structured as follows. Section 2 discusses the related work. Section 3 introduces the tools used for

DOI reference number:10.18293/SEKE2019-177

¹<http://www.openconnectivity.org>

the IoT application. Specifically, the Calvin framework that supported the entire implementation, the MQTT protocol and the OCF “consortium” are described. Section 4 details the proposed architecture and the dynamically controlled smart home application. The implementation and validation of the proposed system are presented in Section 4. Finally, conclusions and future works are presented in Section 5.

2. Related Work

Several studies were executed to design, implement and control IoT systems. Konduru et al [11] presents a study that identifies challenges and solutions considering interoperability of devices. Tools such as Google Weaver, IoTivity, AllJoyn and Apple Home Kit are analysed. Google Weaver and the Apple Home Kit are focused on solving interoperability only for pre-defined devices. While AllJoyn is a framework that together with IoTivity are part of the Open Connectivity Foundation (OCF).

Belsa et al [3] also address the interoperability problem, but with a different approach. IoT platforms are integrated through a flow-based model. The proposed architecture is based on the Node-RED² middleware. However, unlike Calvin, that uses a distributed hash table (DHTs), it does not support the development of applications with distributed systems. [20], also uses Node-RED to design, deploy and control devices remotely in a smart laboratory.

Finally, [12] implements a Healthcare resource model using OCF and IoTivity platform. In this work, two OCF standards are used, the OIC Healthcare Resource and the OIC Healthcare Device. Some of these resource are blood glucose, body metrics, heart rate and oxygen saturation sensors. In which, each resource has a schema and RAML.

3 Basic Concepts

In this section we present the main technologies used in this work.

3.1 Calvin

Calvin is an open source framework aiming to ease the development of IoT applications in distributed systems. It is based on a data flow programming methodology through an actor model [14, 16]. The paradigm of data flow programming is present as an actor model. An actor is a software component that models functions, devices, services or some type of computing in the form of an object. In Calvin, tokens are used to establish communication between actors. These tokens are created when the input actors captures and processes data. Data is then transformed into resources, which in this environment are considered tokens, that will be consumed by other actors [18].

²Node-RED: <https://nodered.org/>

The framework has several standard actors and sensors responsible for input/output of data, media and network access (including HTTP, TCP and MQTT), among others [21]. New actors can be implemented in Python and the dataflow between them is expressed using a declarative language called CalvinScript [19]. Because of the ease of developing actors, Calvin simplifies the implementation of several new services based on this type of model.

To develop an application in Calvin it is necessary to follow a cycle consisting of four phases: describe, connect, deploy and manage. Firstly, the developer will describe how a task is executed by each actor. Secondly, it is necessary to make the connections between them. This can be done using the Calvin GUI. This tool lists the available actors and allows to connect them through their inputs and outputs. When performing this, the CalvinScript is automatically generated. Figure 1 shows a detail of Calvin’s graphical interface. After creating connections, the application is already ready to deploy. Finally, during the management phase, modifications can be made in the application and Calvin executes the deployment [14, 16].

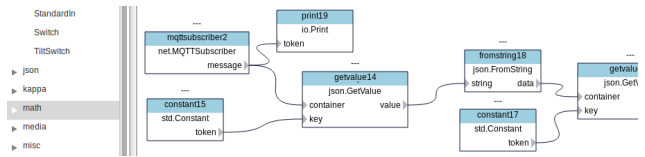


Figure 1. Calvin GUI.

In Calvin’s architecture there is a layer called runtime where actors are executed. It has two sub-layers, platform dependent and platform independent. The first allows communication between several runtimes through the most varied communication protocols, such as WiFi and Bluetooth. The second, allows the execution of several actors at one runtime only. Calvin GUI works on that layer.

3.2 MQTT

Message Queue Telemetry Transport (MQTT) is an application layer protocol that centralizes the sending and receiving of data from applications using a wireless sensor network, enabling the communication between devices with limited power source. This architecture is based on a Publisher/Subscriber system. Publishers are responsible for sending data collected through sensors, while subscribers consumes the data that was collected [9].

A Broker is used to coordinate the sending and receiving of data in order to ensure that the communication will be reliably. The Broker receives the data collected and classifies the data as topics. The data is then sent to devices interested in a specific topic. The broker also ensures that the data will be received only by the subscribers who will consume them. To provide safety and quality of data com-

munication, MQTT uses encryption and login to guarantee a minimum level of quality of service (QoS) [13].

The MQTT also allows the creation of a development environment for IoT applications. This is due to its high integration capability with several IoT development platforms, as it enables the development of services using various programming languages, such as Python, Java, JavaScript, PHP, Ruby and C. Also, MQTT allows extending these services to mobile platform such iOS and Android [6].

3.3 OCF and oneIoTa

Open Connectivity Foundation³ (OCF), is an IoT consortium that aims the specifications, open source and certifications needed for the development of an IoT environment with interoperability between devices that acts as OCF Clients and Servers [4, 17]. The exchange of information between devices uses the JSON format, following the specifications for identifying them and their connected resource, which makes it easier to validate the data structure [5]. To describe OCF services the specifications of the resources are made using a descriptive language of RESTful APIs, the RAML. To ease the implementation of the OCF data model, the consortium has developed the open tool oneIoTa⁴ that has several examples of type RAML, Swagger and JSON Schema models. Users can also create their own templates that, if approved, are stored in a GitHub repository.

4 Proposed Solution

As stated before, we are motivated by the following aspects: interoperability; remote access and dynamic configurations. Interoperability is a critical problem in IoT. Application developers should consider providing services to all clients regardless of the hardware platform specifications they use. Integration with different communication devices is necessary so that new functions can be added to the application without compromising existing functions or even losing them [1].

Devices and appliances must also allow remote access in order to be monitored and controlled through a computer or a smartphone [23]. In security applications remote access is indispensable because it allows the user to monitor their home when they are traveling for example.

Finally, IoT applications must allow dynamic configuration. Most of IoT applications developed today were created to perform pre-defined tasks, for that reason they are static systems limited to certain actions. Dynamic configuration makes the system more elastic and comprehensive because the user can configure the applications to handle devices and sensors that will be added later. In addition,

new monitoring and control applications can be created and modified at the users discretion.

To achieve the above goals and to facilitate the implementation of the system, we are looking for an IoT middleware that provides the best possible solution. The existing IoT architectures are divided into three types of classes: service-based; cloud-based; and actor-based. Service-based architectures and actors provide interoperability, they support a specific programming model or device abstraction. However, the service-based model provides limited functionality for the user when it comes to integrating with other applications or even interpreting data. The cloud-based model provides interoperability through specific standards, which is not desired, and the middleware can stop if the cloud provider terminates the service [15]. For these reasons, the actor-based model is the best solution for the smart home system we propose.

An actor-based architecture provides a better way of dealing with large-scale IoT devices, since middleware can be deployed across all layers of the architecture, so devices can perform actions where it is more adequate. Therefore, an actor-based middleware, such as Calvin and Node-RED, is a good choice in applications involving a large number of “things”. In addition to having features such as interoperability, security, and privacy, where users can choose the form and location where the data will be stored [15].

The proposed architecture involves a variety of devices and technologies. The first part consists of sensors, for example temperature sensors, present in wireless nodes. These nodes are connected to a gateway unit which, in turn, is responsible for sending the sensor data, through messages, to the MQTT Broker. The gateway must then be equipped with the Calvin framework with a Publish application that sends the messages with a specific topic for the sensor identification. Calvin Constrained is a good choice in this case due to the limitation of some devices.

In Figure 2 a representation of the logical diagram of the proposed system can be seen. This diagram provides a view of the connection and communication between the architecture elements. Each component behaves as follows:

1. The Middleware receives the data sent by the sensors with its information, and modifies that data and transforms it into a JSON object. This object is a sequence of key/value pairs. A key must be a string, and the value must be a JSON base type;
2. The Gateway is a MQTT client that connects to the cloud server through a TCP/IP connection. Once connected the gateway can publish the data of sensors and devices for the broker to distribute to the controller and also subscribe to the data published by the broker to the devices. The broker acts as an intermediary between the gateway and the controller.
3. The Controller application subscribes to the sensors

³OCF: <https://openconnectivity.org/>

⁴oneIoTa: <https://oneiota.org/>

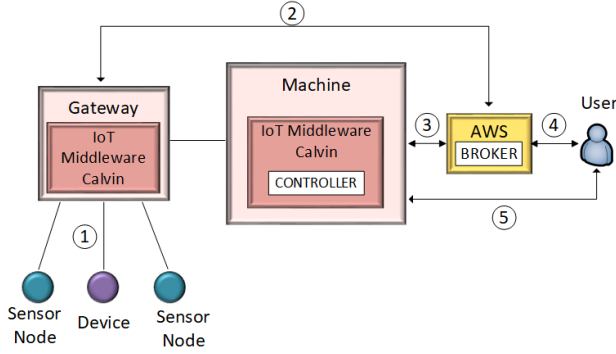


Figure 2. Architecture of the proposed system.

and devices data coming from the Gateway through the cloud, performs the control action and publishes the new data to the device;

4. The user has direct access to the cloud server through the platform on the Internet, and can manage the Broker data;
5. The user can access the application controller directly from the local machine or via remote access.

In order to exchange MQTT messages, a Broker, which is implemented on a cloud platform, is required. There are several free and paid cloud platforms available with support for gateways, services, and application protocols such as REST, COAP, XMPP, and MQTT [1]. Eclipse Mosquitto was chosen for the application because it is a Broker that provides a lightweight server implementation of the MQTT protocol and, moreover, it is open source (EPL/EDL licensed) [7, 8].

Finally, the control module provides the system-wide management through dynamic configuration. The application is implemented in Calvin GUI, which is a web-based GUI for Calvin application development. To access the GUI platform the user must have the Calvin runtime running on a local machine or on a machine on the same network. The dynamic configuration of this application is done in a simple way, given the ease in changing the actors and their data flows in the GUI platform. Therefore, you can at any time interrupt the application's data flow and modify it according to your needs.

4.1 Validation

A smart home application was implemented using the proposed architecture to validate the proposed solution. Figure 3 provides a visual illustration of the technologies and how they connect with each other. The application consists of a system that collects sensor data and transmits it to

the controller. This data is used to monitor and to control the actuator device dynamically and remotely. The use case consists of performing the temperature control of environments by means of an air conditioning device and sensor nodes.

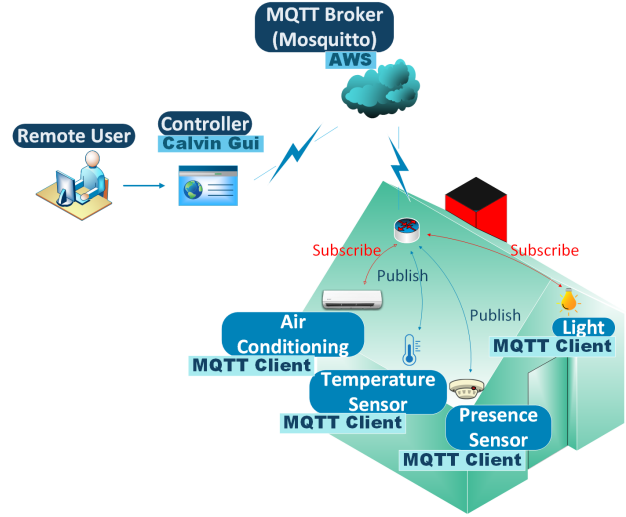


Figure 3. Smart home system overview.

A test environment consisting of four (4) Linux based Virtual Machines running Calvin the framework. Two of these machines were used to simulate an environment of a residence with their devices, i.e. the air conditioners. The other two machines were used to simulate the temperature sensors distributed in the environments. In Figure 4 the sequence of messages of two different simulated environments, living room and room, can be observed.

```
{'topic': 'u/ocf/rt/sala/ar/oic.r.TemperaturaArCondicionado', 'payload': '{
  "rt": ["oic.r.TemperaturaArCondicionado"], "temperatura": 22, "id": "tempe
  ratura"}'}
{'topic': 'u/ocf/rt/sala/ar/oic.r.TemperaturaArCondicionado', 'payload': '{
  "rt": ["oic.r.TemperaturaArCondicionado"], "temperatura": 22, "id": "tempe
  ratura"}'}
{'topic': 'u/ocf/rt/quarto/ar/oic.r.TemperaturaArCondicionado', 'payload': '{
  "rt": ["oic.r.TemperaturaArCondicionado"], "temperatura": 20, "id": "tem
  peratura"}'}
{'topic': 'u/ocf/rt/quarto/ar/oic.r.TemperaturaArCondicionado', 'payload': '{
  "rt": ["oic.r.TemperaturaArCondicionado"], "temperatura": 20, "id": "tem
  peratura"}'}
{'topic': 'u/ocf/rt/quarto/switch/oic.r.SwitchBinary', 'payload': '{"r": "o
  ic.r.SwitchBinary", "id": "switch", "value": 1}'}
{'topic': 'u/ocf/rt/quarto/switch/oic.r.SwitchBinary', 'payload': '{"r": "o
  ic.r.SwitchBinary", "id": "switch", "value": 1}'}
{'topic': 'u/ocf/rt/sala/ar/oic.r.TemperaturaArCondicionado', 'payload': '{
```

Figure 4. Sequence of messages between entities.

To validate the interoperable feature, it was used the oneIoTa specification for OCF devices⁵. Table 1 details the specifications of air conditioner devices. As can be observed in Figure 4, the payload of the devices are in accordance with the recommendations of the OCF, but as a

⁵OCF Device Specification: https://openconnectivity.org/specs/OCF_Device_Specification_v2.0.0.pdf

differentiation between the requirements of the sensor and the air conditioner, the air conditioner was modified to facilitate the understanding of reader. These specifications, as mentioned in 3.3, enables interoperability between devices. OCF defines the resource model that provides consistency among the devices in the home. This model uses the system of resources and devices, whose features exchanged are of various types, in the case of air conditioner has the type binary and the temperature. Each resource type defines a set of properties that are defined using the JSON format.

Table 1. OCF Device Resources.

Device Name	Required Resource name	Required Resource Type	RAML/JSON example
Air Conditioner	Binary Switch	oic.r.switch.binary	{ "rt": ["oic.r.switch.binary"], "id": "unique.example.id", "value": false }
	Temperature	oic.r.temperature	{ "rt": ["oic.r.temperature"], "id": "unique.example.id", "temperature": 20.0, "units": "C", "range": [0.0,100.0] }

To fulfill the remote access feature, Calvin GUI was used to perform updates remotely. The framework allows the control to be remotely carried out on the machine where Calvin is installed or on another machine on same network. In addition, the inclusion of a virtual machine in the cloud, used as a broker running the MQTT server, allows the user to subscribe to a topic of interest. The data published in this topic is sent to those who requested it. In Figure 4 it is possible to identify 3 different topics that the user can sign in. Figure 5 shows more clearly how the message exchange occurred in the application.

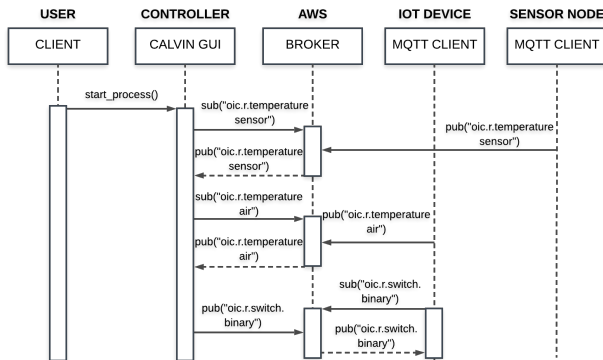


Figure 5. Subscriber topics in JSON format.

The user must initiate the process so that the controller requests the room temperature to the broker. The broker acquires this data from the sensor temperature and sends this value to the controller. The same occurs with the air conditioner. The control will determine whether the air condi-

tioner stays on/off or modifies its state according to the data available and the user specification.

To achieve dynamic configuration, the user must be able to add new devices, modify the control model, and its parameters. Figure 6 shows a small part of the control implemented. Figure 6(a) shows a specific setpoint for the control of the switch of the air conditioner. In Figure 6(b) this setpoint was removed from the control and replaced in the Figure 6(c) (setpoint2). This sequence was executed in runtime, where the controller device was kept running during the changing of setpoints. Also, by the use of a standardized data model (oneIoTa), all remote clients were able to keep receiving updates without any interruption.

5. Conclusions and future work

This article presented an architecture that integrates several technologies, such as IoT protocols and frameworks, to provide a scalable and dynamic environment for the development of services to remotely control and monitoring devices such as, sensors and actuators. The whole architecture was developed following a model based on actors, which enabled a dynamic manipulation of components for IoT applications. As also, using a widely-used protocol, it was possible to establish the communication in a practical and intuitive way through a MQTT broker integrating with cloud services. A JSON based data model (oneIoTa) was used to ensure interoperability between devices. The approach has been successfully developed, as can be verified in its validation procedure. This infrastructure can be adapted to other environments, such as industrial, hospital or business, according to each user's need. In the future, we should include several other devices, providing a means of remote monitoring of these devices and their location in application domains.

References

- [1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.
- [2] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta. Role of middleware for internet of things: A study. *International Journal of Computer Science and Engineering Survey*, 2(3):94–105, 2011.
- [3] A. Belsa, D. Sarabia-Jacome, C. E. Palau, and M. Esteve. Flow-based programming interoperability solution for iot platform applications. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 304–309. IEEE, 2018.
- [4] S. Cavalieri, M. G. Salafia, and M. S. Scroppo. Real-

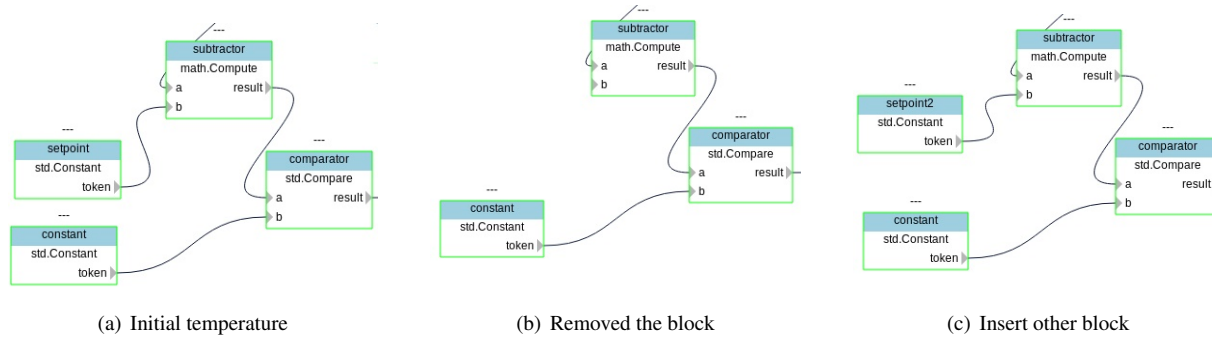


Figure 6. Dynamic Control of Air Conditioner temperature.

ising interoperability between opc ua and ocf. *IEEE Access*, 6:69342–69357, 2018.

- [5] S. Cavalieri and M. S. Scroppo. A proposal to make ocf and opc ua interoperable. In *Proceedings of ICIT*, volume 2018, 2018.
- [6] M. Collina, G. E. Corazza, and A. Vanelli-Coralli. Introducing the qest broker: Scaling the iot by bridging mqtt and rest. In *Personal indoor and mobile radio communications (pimrc), 2012 ieee 23rd international symposium on*, pages 36–41. IEEE, 2012.
- [7] I. Eclipse Foundation. *Eclipse Mosquitto: An open source MQTT broker*, 2018 (accessed August 23, 2018). <https://mosquitto.org/>.
- [8] I. Eclipse Foundation. *Eclipse Mosquitto*, 2018 (accessed July 3, 2018). "<http://projects.eclipse.org/projects/technology.mosquitto>".
- [9] U. Hunkeler, H. L. Truong, and A. Stanford-Clark. Mqtt-sa publish/subscribe protocol for wireless sensor networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, pages 791–798. IEEE, 2008.
- [10] R. Khan, S. U. Khan, R. Zaheer, and S. Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 257–260. IEEE, 2012.
- [11] V. R. Konduru and M. R. Bharamagoudra. Challenges and solutions of interoperability on iot: How far have we come in resolving the iot interoperability issues. In *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, pages 572–576. IEEE, 2017.
- [12] J.-C. Lee, J.-H. Jeon, and S.-H. Kim. Design and implementation of healthcare resource model on iotivity platform. In *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 887–891. IEEE, 2016.
- [13] S. Lee, H. Kim, D.-k. Hong, and H. Ju. Correlation analysis of mqtt loss and delay according to qos level. In *Information Networking (ICOIN), 2013 International Conference on*, pages 714–717. IEEE, 2013.
- [14] A. Najafi Nassab. Mobile devices in the distributed iot platform calvin. 2017.
- [15] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, 4(1):1–20, 2017.
- [16] T. Nilsson. Authorization aspects of the distributed dataflow-oriented iot framework calvin. 2016.
- [17] S. Park. Ocf: A new open iot consortium. In *Advanced Information Networking and Applications Workshops (WAINA), 2017 31st International Conference on*, pages 356–359. IEEE, 2017.
- [18] P. Persson and O. Angelsmark. Calvin—merging cloud and iot. *Procedia Computer Science*, 52:210–217, 2015.
- [19] P. Persson and O. Angelsmark. Kappa: serverless iot deployment. In *Proceedings of the 2nd International Workshop on Serverless Computing*, pages 16–21. ACM, 2017.
- [20] M. Poongothai, P. M. Subramanian, and A. Rajeswari. Design and implementation of iot based smart laboratory. In *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 169–173. IEEE, 2018.
- [21] E. Research. calvin-base. <https://www.github.com/EricssonResearch/calvin-base>, 2018.
- [22] S.-H. Yang. Internet of things. In *Wireless Sensor Networks*, pages 247–261. Springer, 2014.
- [23] M. Yun and B. Yuxin. Research on the architecture and key technology of internet of things (iot) applied on smart grid. In *Advances in Energy Engineering (ICAEE), 2010 International Conference on*, pages 69–72. IEEE, 2010.
- [24] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

Reverse Engineering Behavioural Models of IoT Devices

Sébastien Salva

LIMOS - UMR CNRS 6158

University Clermont Auvergne, France

email: sebastien.salva@uca.fr

Elliott Blot

LIMOS - UMR CNRS 6158

University Clermont Auvergne, France

email: elliott.blot@uca.fr

Abstract—This paper addresses the problem of recovering behavioural models from IoT devices in order to help engineers understand how they are functioning and audit them. We present a model learning approach called ASSESS, which takes as inputs execution traces collected from IoT devices and generates models called systems of Labelled Transition Systems (LTSs). ASSESS generates as many LTSs as components integrated and identified into a device. The approach is specialised to IoT devices as it takes into account two architectures often used to integrate components with this kind of system (cyclic functioning, loosely-coupled or decoupled architectures). We experimented the approach on two IoT devices and an IoT gateway to evaluate the model conciseness and the approach efficiency.

Keywords—Reverse engineering; IoT; Model learning; Passive learning.

I. INTRODUCTION

Internet connected devices, and especially Internet of Things (IoT), belong to the digital transformation trends proposed by industrial experts or advisory firms for several years. The IoT, which we consider as a network of smart embedded devices connected to the Internet, is indeed a broad-based concept, transforming several uses from consumer devices to large-scale manufacturing. However, many customers and companies prefer staying away from the IoT hype because of the issues related to privacy and more generally to security. It is indeed manifest that IoT devices have to be audited before using them, in particular in the industry or in healthcare. Many companies chose to outsource the IoT development for saving costs, hence the IoT audit is rather done after the development. It is often carried out from the source code or from devices seen as black boxes. A common solution to help audit such devices is to apply a reverse engineering process, which is usually done by hands. From a black-box, this process is required to understand how devices are functioning. Besides, it helps document the behaviours of IoT devices or IoT networks, and may serve to detect bugs or security issues.

In the literature, some papers dealing with the reverse engineering of IoT devices have been published recently [1], [2]. These approaches recover critical information or detect privacy issues from source codes, firmwares or chips.

This paper proposes another approach called ASSESS (Analysis, Extraction, Separation, Synchronisation) to recover behavioural models from IoT devices. Our approach, which is based on the model learning concept, takes execution traces collected from IoT devices and generates models called systems of LTSs (Labelled Transition System). Model learning approaches [3], [4], [5], [6], [7], [8] have proven to be valuable for retro-engineering models that can be exploited in several software engineering steps. Our approach advances the state of the art in these two points.

- It is specialised to IoT devices in the sense that their general functioning is considered while the model generation. We define an IoT device as an embedded device integrating several components and running in a cyclic way [9]. Several works focused on the component architectures of embedded devices, i.e. on how to compose them efficiently. It is often advised to use a loosely-coupled architecture [10], where components remain autonomous and allow middleware software to manage internal communication between them. With this kind of architecture, components are synchronised together. However, we also observed that IoT devices may also have a decoupled architecture, where the components operate independently. We consider both architectures for the model generation.
- Most of the model learning algorithms build one big model for a given system. Such models may quickly become uninterpretable. We focused on this problem and laid the first stone of the approach in [11]. Our approach builds as many LTSs as components detected in logs collected from an IoT device. From these messages, our approach is able to build traces and infer systems of LTSs. Two strategies, which refer to the previous IoT device architectures, are proposed to synchronise LTSs together to form a complete model. The present approach target specific system, IoT devices, and relax some assumption with the modification of algorithms.

We have implemented a prototype tool to experiment our algorithms and appraise their benefits. We provide a preliminary evaluation in the paper made on two IoT devices. Besides, this experimentation also shows that ASSESS may be applied on an IoT gateway to recover a model

expressing the behaviours of an IoT network, i.e. of the devices communicating with this gateway.

The paper is organized as follows: we recall some definitions about the LTS model in Section II. Our approach is presented in Section III. The next section shows some results of our experimentation. Section V summarises our contributions and draws some perspectives for future work.

II. THE LTS MODEL

We express the behaviours of components with Labelled Transition Systems (LTS) as defined in [12]. This model is defined in terms of states and transitions labelled by actions, taken from a general action set \mathcal{L} , which expresses what happens. τ is a special symbol encoding an internal (silent) action; it is common to denote the set $\mathcal{L} \cup \tau$ by \mathcal{L}_τ .

Definition 1 (LTS) A Labelled Transition System (LTS) is a 4-tuple $\langle Q, q_0, \Sigma, \rightarrow \rangle$ where:

- Q is a finite set of states; q_0 is the initial state;
- $\Sigma \cup \{\tau\} \subseteq \mathcal{L}_\tau$ is the finite set of actions, with τ the internal (unobservable) action;
- $\rightarrow \subseteq Q \times \Sigma \cup \{\tau\} \times Q$ is a finite set of transitions. A transition (q, a, q') is also denoted $q \xrightarrow{a} q'$.

We use the generalised transition relation \rightarrow to represent LTS paths: $q \xrightarrow{a_1 \dots a_n} q' =_{def} \exists q_0 \dots q_n, q = q_0 \xrightarrow{a_1} q_1 \dots q_{n-1} \xrightarrow{a_n} q_n = q'$. We also use the following notations on action sequences. The concatenation of two action sequences $seq_1, seq_2 \in \mathcal{L}_\tau^*$ is denoted $seq_1.seq_2$. ϵ denotes the empty sequence. A trace is a finite sequence of observable actions in \mathcal{L}^* .

To better match the functioning of IoT devices, we assume that an action has the form $a(\alpha)$ with a a label and α an assignment of parameters in P , with P the set of parameter assignments. For example, $switch(id := 115, cmd := on)$ is made up of the label "switch" followed by the assignment $(id := 115, cmd := on)$ of two parameters.

The use of LTSs allows to exploit the definitions related to the LTS composition. The integration of two components C_1 and C_2 , modelled with LTSs, is often defined by two operations in the literature. The first one is the parallel composition of C_1 and C_2 denoted $C_1 \parallel C_2$, which synchronises their shared actions, also called *synchronisation actions* (the rest must happen independently). This composition is often followed by the hiding of the communications between C_1 and C_2 to express that only the communications with the environment are observable. This operation is defined by the relation $hide\ S\ in\ C_1 \parallel C_2$ with S a set of actions. We refer to [13] for the definitions of these two LTS operators.

This principle of LTS composition leads to a model called system of LTSs, which describes a component-based system:

Definition 2 (System of LTSs) A system of LTSs SC is the couple $\langle S, C \rangle$ with $C = \{C_1, \dots, C_n\}$ a non empty set of LTSs, and S a set of synchronisation actions.

III. THE ASSESS APPROACH

This section presents our model learning approach, which aims at inferring system of LTSs from messages given by an IoT device. The later is seen as a black box and integrates components by means of a loosely coupled or a decoupled architecture. We assume that the components produce messages or logs which include component identifiers, i.e. parameter assignments allowing to identify components. However we consider that the component calls are hidden. This is usually the case with IoT devices integrating several sensors. Furthermore, the messages have to include timestamps for ordering them. A logical clock mechanism may be required to add timestamps in logs.

The list of messages is initially translated into a set of execution traces with our tool TFormat¹. This one starts by filtering and formatting raw messages into actions by means of regular expressions. Then, the tool analyses the timestamps of every pair of successive actions and computes means of time intervals. It searches for gaps between actions (distinctive longer durations), which are usually observed when an execution trace ends and another one begins. The time gap detection is used for the trace extraction. We denote the trace set $Traces(SUL)$ and assume that a trace has the form $a_1(\alpha_1) \dots a_k(\alpha_k)$.

The model generation is performed by three steps called "Trace Extraction", "LTS Generation", and "LTS Synchronisation". The last step proposes two LTS generation strategies called "LTS Loose-coupling" and "LTS Decoupling". These steps are illustrated with the example of Figure 1. In the first step, the traces of $Traces(SUL)$ are analysed to detect component calls by covering the component identifiers found in actions. The example of Figure 1 lists 3 traces that capture the behaviours of two components ($id:=1, id:=3$), which call other components. The component calls are here detected whenever a new identifier is found ($id:=2, id:=3$). In a trace, the action sequences having different identifiers are extracted and replaced by synchronisation actions of the form $call(id)$ and $return(id)$ to express component calls, with id an identifier referring to a component. Next, the resulting traces are partitioned to gather the traces having the same identifier. We obtain 3 trace sets in our example of Figure 1.

The step "LTS Generation" transforms each previous trace set into a LTS. In this step, we take into account the general functioning of the IoT devices, which are usually designed to perform actions in a cyclic way. The traces are hence transformed into cyclic LTS paths, the later being joined on an initial state. Once every trace set is transformed into a LTS, we obtain a first system of LTSs $SC = \langle S, C \rangle$ with C the set of LTSs and S the set of synchronisation actions.

The last step transforms this system of LTSs to produce more general models with respect to the nature of the IoT devices. As stated earlier, we consider that these devices

¹<https://github.com/sasa27/TFormat>

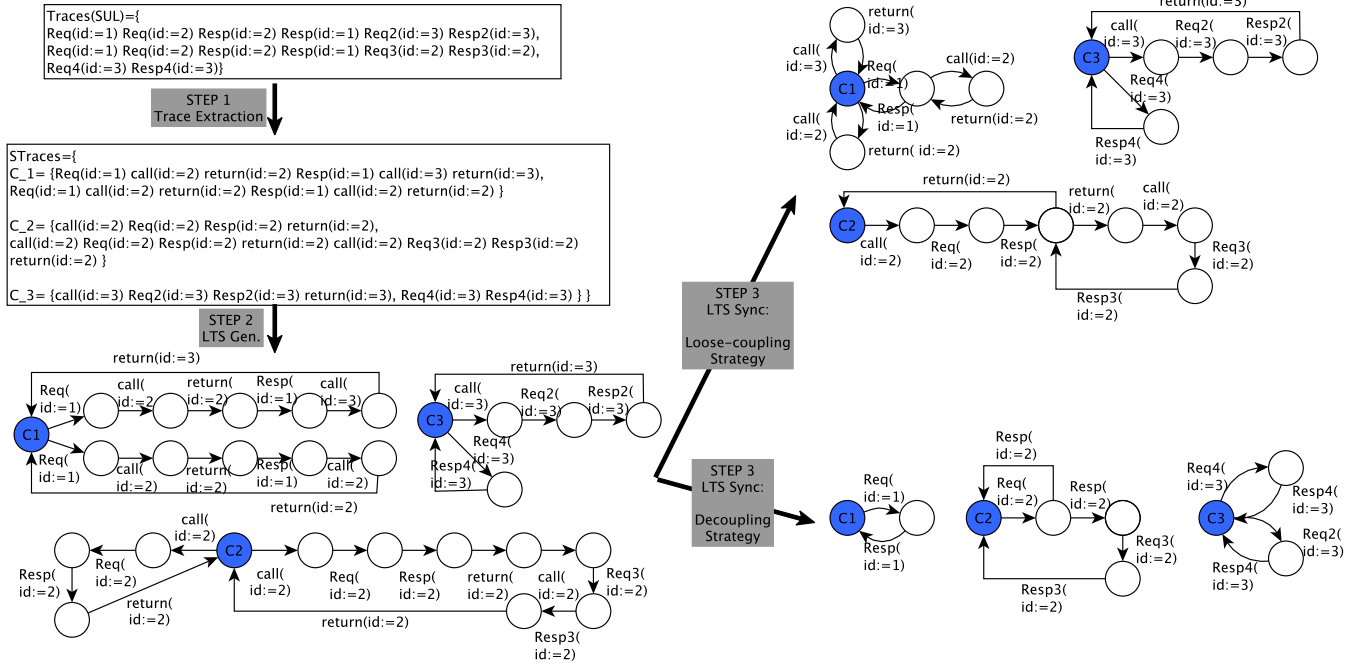


Figure 1: The ASSESS approach overview

may integrate loosely-coupled or decoupled components. The strategy “LTS Loose-coupling” builds a system of LTSs $SC1$ such that $SC1$ allows repetitive calls of components, which are synchronised together. This is materialised by replacing the sequences $q_1 \xrightarrow{call(id) \ return(id)} q_2$ with loops. Then, we apply the kTail algorithm [3]. kTail is a well-known approach that merges the (equivalent) states having the same k-future, i.e. the same event sequences having the maximum length k . We obtain three LTSs in Figure 1(right-top side) expressing components that call each other.

The strategy “LTS Decoupling” produces another system of LTSs $SC2$ from SC to express the behaviours of independent components. The synchronised actions are removed from the LTSs of SC . Then, the kTail algorithm is applied. We obtain three LTSs expressing autonomous components. Now, we detail these steps below.

A. Step 1: Trace Extraction

This step covers the traces of $Traces(SUL)$ and the identifiers included in actions to detect implicit component calls and to gather the traces related to each component in separate trace sets. The following definition formalises the notion of component identification:

Definition 3 (Component identification) Let $a_1(\alpha_1)$ be an action of \mathcal{L} . The component identifier of $a_1(\alpha_1)$ is given by the mapping $ID : \mathcal{L} \rightarrow P$, which gives the parameter assignment α' found in α_1 that identifies the component producing the action $a_1(\alpha_1)$.

The component identifier of a sequence $a_1(\alpha_1)a_2(\alpha_2) \dots a_k(\alpha_k)$ is given by the mapping $ID_s : \mathcal{L}^* \rightarrow P$. $ID_s(a_1(\alpha_1)a_2(\alpha_2) \dots a_k(\alpha_k)) =_{def} \begin{cases} \alpha' \text{ iff } \forall a_i \notin \{call, return\} : ID(a_i(\alpha_i)) = \alpha' (1 \leq i < k) \\ \{\} \text{ otherwise.} \end{cases}$

For simplicity, we denote the mapping ID_s by ID in the remainder of the paper.

Algorithm 1: Component Trace Detection

```

input :  $Traces(SUL)$ 
output :  $STraces$ 
1  $Traces := \{\}$ ;
2 foreach  $t = a_1(\alpha_1)a_2(\alpha_2) \dots a_k(\alpha_k) \in Traces(SUL)$  do
3    $id := ID(a_1(\alpha_1)); T := \{\}$ ;
4    $T := Extract(t, T, id)$ ;
5    $Traces := Traces \cup T$ ;
6  $STraces := GroupById(Traces)$ ;
7 return  $STraces$ ;

```

The Trace Extraction step is implemented with Algorithm 1, and its two procedures *Extract* and *GroupById*. The algorithm covers every trace t of $Traces(SUL)$, extracts the identifier id of the first running component found in the first action of t and calls the procedure *Extract*. The latter takes t , id and a set T used to store new traces. *Extract* potentially splits t into several traces, each having one non empty component identifier. Then, the procedure *GroupById* partitions all the traces given by *Extract* and returns the set $STraces = \{C_1, C_2, \dots, C_n\}$ such that the traces of a set C_i exhibit the behaviour of one component only. The procedure *Extract*($t = a_1(\alpha_1)a_2(\alpha_2) \dots a_k(\alpha_k)$, T, id) is given in Algorithm 2. It covers the component identifiers in the actions of t to detect component calls. While covering the actions of t , if an identifier n different from *newid* (first identifier of the current trace) is found (line 6), we assume that a new component has been called by the current one. In this case, the procedure searches for the sequence $a_{i+1}(\alpha_{i+1}) \dots a_{j-1}(\alpha_{j-1})$ composed of actions having identifiers different from *newid*. This sequence is extracted and replaced by the synchronisation actions

Algorithm 2: Procedure Extract

```

1 Procedure Extract( $t = a_1(\alpha_1)a_2(\alpha_2) \dots a_k(\alpha_k), T, id$ ):  $T$  is
2    $newid := Identifier(a_1(\alpha_1));$ 
3    $t' := a_1(\alpha_1); a_{k+1}(\alpha_{k+1}) = \epsilon; i := 1;$ 
4   while  $i < k$  do
5      $n := ID(a_{i+1}(\alpha_{i+1}));$ 
6     if  $n == newid$  then
7        $t' := t'.a_{i+1}(\alpha_{i+1});$ 
8        $j := i + 1;$ 
9     else
10      find smallest  $j > i$  such that  $ID(a_j(\alpha_j)) == newid$ 
11      or  $j := k + 1;$ 
12       $t' := t'.call(n).return(n).a_j(\alpha_j);$ 
13      if  $(j - i) > 2$  then
14         $Extract(a_{i+1}(\alpha_{i+1}) \dots a_{j-1}(\alpha_{j-1}), T, id);$ 
15      else
16         $t_n := call(n).a_{i+1}(\alpha_{i+1}).return(n);$ 
17        if  $\exists t_2 \in T : ID(t_2) == n$  then
18           $t_n := t_2.t_n; T := T \setminus \{t_2\};$ 
19         $T := T \cup \{t_n\};$ 
20       $i := j;$ 
21   if  $newid \neq id$  then
22      $t' := call(newid).t'.return(newid);$ 
23   if  $\exists t_2 \in T : ID(t_2) == newid$  then
24      $t' := t_2.t'; T := T \setminus \{t_2\};$ 
25    $T := T \cup \{t'\};$ 
26   return  $T;$ 

```

$call(n).return(n)$, which model the call of a component C_n . If the extracted sequence has more than one action, the procedure *Extract* is recursively called (line 13). Otherwise, it builds a trace t_n composed of the action $a_{i+1}(\alpha_{i+1})$ surrounded by synchronisation actions. If there exists a trace t_2 in T having the identifier n , t_n is concatenated to t_2 . t_n is added to the trace set T . Once the trace t is covered, we obtain a new trace t' including synchronisation actions. The procedure *Extract* eventually checks whether t' has to be completed to express that this trace was produced by a component called by another one: if the identifier of t' is different from the identifier id given as input (line 20) then the trace t' is surrounded with $call(idnew)$ and $return(idnew)$. Finally, if there exists a trace t_2 in T having the component identifier $idnew$, then t' is concatenated to t_2 . The final trace t' is added to T .

The procedure *GroupById(Traces) : STraces*, partitions the trace set *Traces* in such a way that every subset holds traces sharing the same non empty component identifier. We partition *Trace* by defining the trace equivalence relation \sim_{id} and by extracting the equivalences classes of *Trace* for \sim_{id} . Let \sim_{id} on \mathcal{L}^* be given by $\forall seq_1, seq_2 \in \mathcal{L}^*, seq_1 \sim_{id} seq_2$ iff $ID(seq_1) = ID(seq_2)$. The procedure *GroupById* returns the partition $STraces = Trace / \sim_{id}$.

B. Step 2: LTS Generation

At this stage, *STraces* gathers n subsets with n the number of component identifiers found in the traces of *Traces*(SUL). These subsets of traces are now transformed into LTSs. Intuitively, given T_1 in *STraces*, a trace of T_1 is

lifted to the level of a LTS cyclic path. The LTS is obtained after joining the paths by means of a disjoint union on the state q_0 :

Definition 4 (LTS inference) Let $T_1 \in STraces$ be a trace set. The LTS C_1 expressing the behaviours found in T_1 is the tuple $\langle Q, q_0, \Sigma, \rightarrow \rangle$ where q_0 is the initial state, and Q, Σ, \rightarrow are defined by the following rule:

$$\frac{t = a_1(\alpha_1) \dots a_k(\alpha_k), id = ID(t)}{q_0 \xrightarrow{a_1(\alpha_1)} q_{id1} \dots q_{idk-1} \xrightarrow{a_k(\alpha_k)} q_0}$$

Once the LTS generation is completed, we obtain a first system of LTSs $SC = \langle S, C \rangle$ with C the set of LTSs derived from *STraces* and S the set of synchronized actions.

C. Step 3: LTS Synchronization

Algorithm 3: LTS Synchronisation Strategies

```

1 Procedure Loose-coupling( $SC = \langle S, \{C_1, C_2, \dots, C_n\} \rangle$ ):  $SC_1$  is
2   foreach  $C_i = \langle Q, q_0, \Sigma, \rightarrow \rangle \in C$  do
3     foreach  $q_1 \xrightarrow{call(\sigma).return(\sigma)} q_2$  do
4       merge  $q_1$  and  $q_2$ ;
5        $C'_i := kTail(k = 2, C_i);$ 
6   return  $\langle S, \{C'_1, C'_2, \dots, C'_n\} \rangle$ 
7 Procedure Decoupling( $SC = \langle S, \{C_1, C_2, \dots, C_n\} \rangle$ ):  $SC_2$  is
8   foreach  $C_i = \langle Q, q_0, \Sigma, \rightarrow \rangle \in C$  do
9      $C_i := \text{hide } S \text{ in } C_i;$ 
10     $C_i := \tau\text{-reduce } C_i;$ 
11     $C'_i := kTail(k = 2, C_i);$ 
12  return  $\langle S, \{C'_1, C'_2, \dots, C'_n\} \rangle$ 

```

This last step proposes two strategies to synchronise the LTSs of SC with regard to the architecture considered to integrate components together. Both strategies are implemented in Algorithm 3 with two procedures.

The strategy “LTS Loose-coupling” builds a new system of LTSs SC_1 from SC and keeps the transitions carrying synchronised actions. This strategy allows repetitive calls of components but also makes these calls optional by replacing the transition sequences of the form $q \xrightarrow{call(\sigma).return(\sigma)} q'$ by loops (lines 3,4).

The strategy “LTS Decoupling” gives another system of LTSs SC_2 from SC by firstly hiding the synchronisation actions. The operator $\text{hide } S \text{ in } C_i$ transforms the transitions of C_i by replacing the actions of S with the non observable action τ . We then reduce C_i by removing the transition labelled by τ . Several algorithms are proposed in the literature to perform this LTS reduction with respect to a given LTS equivalence relation. However, as the LTSs generated by Step 2 have a simple structure (only one outgoing transition per state), we propose a lightweight LTS reduction operation denoted τ -reduction:

Definition 5 (τ -reduction) Let $C_1 = \langle Q_1, q_{01}, \Sigma, \rightarrow_1 \rangle$ be a LTS. τ -reduction $C_1 =_{def} \langle Q_2, q_{02}, \Sigma, \rightarrow_2 \rangle$ where $Q_2, q_{02}, \rightarrow_2$ are the minimal sets satisfying the following inference rules:

$$\begin{array}{ccc} \frac{q_1 \xrightarrow{a(\alpha)} q_2}{q_1 \xrightarrow{a(\alpha)}_2 q_2} & \frac{q_1 \xrightarrow{a(\alpha)} q_2 \xrightarrow{\tau \dots \tau} q_3}{q_1 \xrightarrow{a(\alpha)}_2 (q_2 q_3)} & \frac{q_1 \xrightarrow{\tau \dots \tau} q_2 \xrightarrow{a(\alpha)} q_3}{(q_1 q_2) \xrightarrow{a(\alpha)}_2 q_2} \end{array}$$

kTail is finally applied on the LTSs achieved by both strategies. We use $k = 2$ as recommended in [6].

Both systems of LTSs SC_1 and SC_2 offer different points of view. With SC_1 , the component calls are explicitly given, which offers the possibility of extracting a dependency graph of components showing how the components are hierarchically organised. With the system of LTSs SC_2 , as the transitions carrying synchronised actions are removed, the parallel composition of the LTSs expresses the behaviours of asynchronous and autonomous components, which hence produce actions independently of the others. As it is illustrated in Figure 1, the second strategy returns more compact and general models.

IV. PRELIMINARY EVALUATION

We have implemented our approach in a tool, with which we began a first evaluation to answer to these two questions:

- RQ₁: can ASSESS extract more concise and readable models than the ones generated by kTail?
- RQ₂: how long does ASSESS take to generate models?

Setup: we applied ASSESS on two IoT devices and one IoT gateway. The first device (*exp.1*) is a smart thermostat controlling heat-pumps via infra-red, composed of 4 components (a Web server, two sensors, and a component that manages the heating mode). The second device (*exp.2*) is a Wifi IP camera that integrates 5 components. The IoT gateway (*exp.3*) was interconnected to 8 autonomous devices, which we consider as components for the experimentation. We collected HTTP messages from these systems and formatted them with our tool TFormat. The results and the tool are available here².

A. Question RQ₁

Procedure: we collected traces for every setup and ran ASSESS with its two strategies. We also ran kTail on the same trace sets for comparison purposes. Then, we measured the sizes of the generated models. These are given in Table I. Furthermore, with large trace sets, model learning might return spaghetti-like models, containing an uninterpretable mess of transitions. We compared the generated models to deduce whether ASSESS can significantly help reduce this spaghetti model problem by inferring one model per component.

Exp.	kTail		Loosely-coupled		Loosely-coupled without <i>call</i> and <i>return</i>		Decoupled	
	#states	#trans	#states	#trans	#states	#trans	#states	#trans
<i>exp.1</i>	52	90	116	208	61	118	31	54
<i>exp.2</i>	92	186	172	346	80	193	36	76
<i>exp.3</i>	349	419	426	552	362	439	310	339

Table I: Size of the LTSs obtained with kTail and ASSESS.

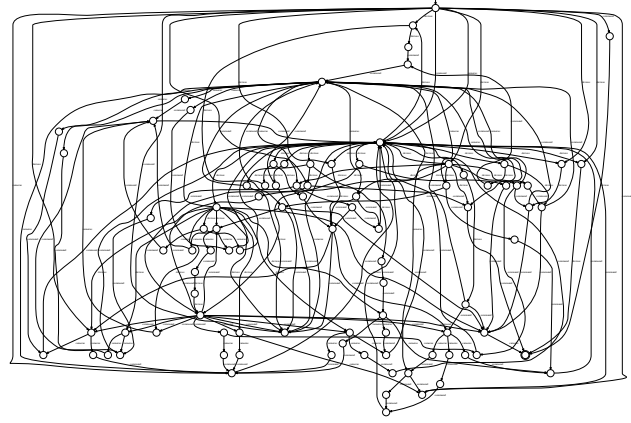


Figure 2: Overview of the models generated with kTail (*exp.3*)

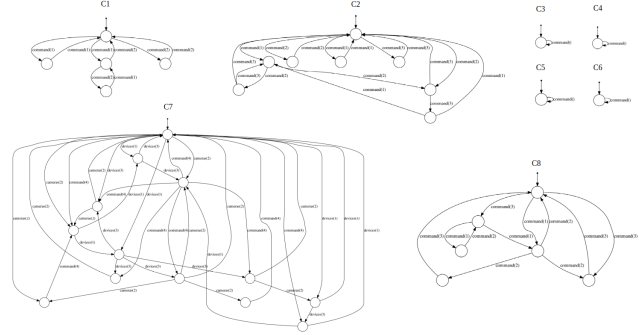


Figure 3: Overview of the models generated with ASSESS (*exp.3*)

Results: Table I shows that we obtain larger transition sets with the "LTS Loose-coupling" strategy. In average, the state number is increased by 77.33% in comparison to the results of kTail. This is due to the addition of transitions labelled by synchronisation actions, which show how components interact with one another. If we do not take into account these transitions, we obtain models whose sizes are close to the sizes of the models generated by kTail. With the "LTS Decoupling" strategy, we always obtain more concise models. The state number is reduced on average by 37.33% with this strategy. The state reduction is a consequence of the segmentation of the traces by our algorithm. We infer one LTS for each component, which is easier to reduce with kTail than one big model. Afterwards, we compared the models generated by kTail and ASSESS and manifestly concluded on these experimentations that the systems of LTSs are significantly more interpretable. Figure 2 shows an overview of the "spaghetti"-like model generated by kTail for *exp.3*. This model (even zoomed) is difficult to understand. Figure 3 illustrates the system of LTSs generated by ASSESS (second strategy). We believe that the later is more readable since every component is represented by its own model whose transition set is smaller. Besides, a system of LTSs sounds more adaptable to the user needs. For instance, an undesired component may be concealed to help focus on the others.

²<https://github.com/Elblot/ASSESS>

B. Question RQ₂

Procedure: to investigate RQ₂, we measured the execution times of ASSESS with several trace sets containing 10 to 35000 traces of around 150 events collected from *exp.3*. Experimentations were done on a computer with 1 Intel(R) CPU i5-6500 @ 3.2GHz and 16GB RAM. Figure 4 draws two curves showing the execution times measured with both strategies.

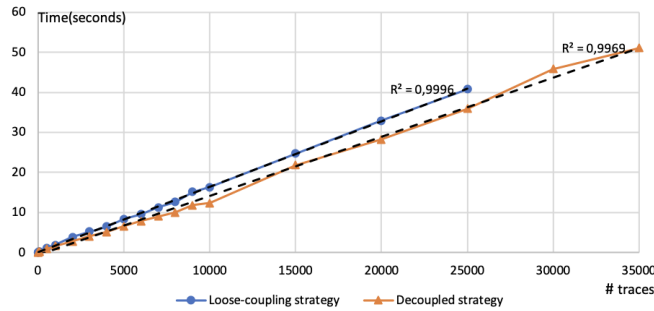


Figure 4: Executions times of ASSESS

Results: Figure 4 shows that ASSESS requires less than 60 seconds to build models with the largest trace set. The tendency curves also confirm that the time complexity of both strategies is linear. Regarding the memory space complexity, we also observed a linear curve; we reached a memory limit between 25000 and 30000 traces (more than 3.5 millions of events) with the Loose-coupling strategy, and between 35000 and 40000 traces (more than 5 millions of events) with the Decoupled strategy. We hence believe that our tool can be used with systems producing a huge amount of messages.

V. CONCLUSION

The increase in IoT technologies popularity holds many benefits, but it is also accompanied by many concerns related to the IoT device reliability and security. Learning models from these devices may serve to audit them. However recovering models usable for inspection is still challenging. So far, most of the learning algorithms build big models and do not take into consideration the IoT device architectures. In this paper, we have presented ASSESS, a model learning method dedicated to IoT devices that recovers systems of LTSs. The method constructs execution traces from messages or logs, and generates LTSs that capture the behaviours of all the components of an IoT device and their synchronisations. Two strategies are proposed to adapt the model generation with regard to the loosely-coupled or decoupled architecture usually used to design embedded devices.

Our future work includes further evaluating ASSESS on other kinds of IoT devices, improving its effectiveness by devising parallel algorithms, and proposing other strategies to better match the available IoT architectures and frameworks.

REFERENCES

- [1] M. Tellez, S. El-Tawab, and M. H. Heydari, "Iot security attacks using reverse engineering methods on wsn applications," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec 2016, pp. 182–187.
- [2] O. Schwartz, Y. Mathov, M. Bohadana, Y. Elovici, and Y. Oren, "Opening pandora's box: Effective techniques for reverse engineering iot devices," in *Smart Card Research and Advanced Applications*, T. Eisenbarth and Y. Tegliala, Eds. Cham: Springer International Publishing, 2018, pp. 1–21.
- [3] A. Biermann and J. Feldman, "On the synthesis of finite-state machines from samples of their behavior," *Computers, IEEE Transactions on*, vol. C-21, no. 6, pp. 592–597, June 1972.
- [4] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," in *Proceedings of the 21st International Conference on Software Engineering*, ser. ICSE '99. New York, NY, USA: ACM, 1999, pp. 213–224.
- [5] K. Meinke and M. Sindhu, "Incremental learning-based testing for reactive systems," in *Tests and Proofs*, ser. Lecture Notes in Computer Science, M. Gogolla and B. Wolff, Eds. Springer Berlin Heidelberg, 2011, vol. 6706, pp. 134–151.
- [6] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic generation of software behavioral models," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE'08. New York, NY, USA: ACM, 2008, pp. 501–510.
- [7] T. Ohmann, M. Herzberg, S. Fiss, A. Halbert, M. Palyart, I. Beschastnikh, and Y. Brun, "Behavioral resource-aware model inference," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. New York, NY, USA: ACM, 2014, pp. 19–30.
- [8] F. Pastore, D. Micucci, and L. Mariani, "Timed k-tail: Automatic inference of timed automata," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, March 2017, pp. 401–411.
- [9] L. Gomes and J. Fernandes, *Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation*, Jan 2009.
- [10] D. S. Stewart, "Designing software components for real-time applications," in *Proceedings of Embedded System Conference*, september 2000.
- [11] S. Salva, E. Blot, and P. Laurençot, "Combining model learning and data analysis to generate models of component-based systems," in *Testing Software and Systems - 30th IFIP WG 6.1 International Conference, ICTSS 2018, Cádiz, Spain, October 1-3, 2018, Proceedings*, 2018, pp. 142–148.
- [12] J. Tretmans, *Model Based Testing with Labelled Transition Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–38. [Online]. Available: https://doi.org/10.1007/978-3-540-78917-8_1
- [13] M. van der Bijl, A. Rensink, and J. Tretmans, "Compositional testing with ioco," in *Formal Approaches to Software Testing*, A. Petrenko and A. Ulrich, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 86–100.

A Resource Management Architecture For Exposing Devices as a Service in the Internet of Things

Carlos Eduardo Pantoja
CEFET/RJ

Universidade Federal Fluminense
pantoja@cefet-rj.br

Heder Dorneles Soares and
José Viterbo and Tielle Alexandre

Universidade Federal Fluminense
hdorneles,viterbo@ic.uff.br, tiellesa@id.uff.br

Arthur Casals and

Amal El-Fallah Seghrouchni
Sorbonne Universités UPMC, LIP6
Universidade de São Paulo (USP)
amal.elfallah@lip6.fr, arthur.casals@usp.br

Abstract—This work proposes an architecture for sharing devices' resources in the Internet of Things providing real sensor data for its users. The main idea is based on the fact that users such as developers and researchers do not always have access to the necessary hardware and resource sharing should impact these persons activities. Taking advantage of the Sensors as a Service model, we propose an architecture where several sensors and actuators can be coupled to environments and they also are represented virtually in a web system becoming available to be consumed by users and platforms. The architecture is composed of three layers and a model representing devices, the cloud, and clients, and how they interact with each other. A study case for testing the whole approach is also presented.

Index Terms—Internet of Things, Embedded Systems, Ubiquitous Computing

I. INTRODUCTION

The number of computing devices used in daily tasks, the internet coverage as well as how to handle data, are increasingly being used to improve people's lives and together with the Internet of Things (IoT), they are the key to the development of a wide range of applications [1]. These large number of components open up relevant issues, such as sharing data in heterogeneous environments once sensors deployment often employs high costs. Then, mechanisms for resource sharing should play a major role in this scenario.

These applications gather data from several sensors and the number of devices can make the project unfeasible. There is an emerging cloud computing model named Sensors-as-a-Service, where users can make public their own sensors' data to be consumed by other clients [2]. However, it is difficult to integrate and deploy such systems due to the heterogeneity of devices and communication technologies [3], [4]. Considering this, middleware for IoT play an important role in these systems since they can deal with the heterogeneity of hardware, data distribution, and communication protocols.

So, the objective of this paper is to propose an architecture for the management of resources and to expose them as a service in an IoT network. This Resource Management Architecture (RMA) is divided into three layers of abstractions where devices are responsible for keeping the real resources independently from the core of RMA, which deals with the virtualization, registering and updating of data from devices.

Then, clients can consume this data as a service using exposed web services that access the virtualization layer. The aim of RMA is to provide an integrated solution to be used in any domain since its general architecture is not bounded to any application provided and to work as a repository of devices where researchers can make their data consumable to other persons who might be interested in them.

The RMA is built over instances of a robust middleware for IoT in its layers, which treats the connectivity and scalability of devices [5]. Besides, we also propose a structure for the development of devices able to self-configuration in the RMA. Finally, the RMA can be accessed by web services or other web solutions. The rest of this paper is structured as follows: in section II, we present the related works; section III presents our proposed architecture and their components; in section IV, it is described the implementation of RMA; section V the RMA is evaluated based on software engineering methods. Finally, conclusions are discussed in section VI.

II. RELATED WORKS

During the last years, several approaches try to deal with shareable resources in different domains. For example, the research in Ambient Intelligent has been searching for technological strategies to improve the people life quality and solve problems caused by population growth in urban areas. A lot of papers explore technologies for improving citizen services and how to add value to public administration in Smart Cities [6]. In these cases, there are data coming from people, social networks, household, organizations (public and privates), and so on. Devices are often used in such approaches however, issues as interoperability, communicability, and how to turn data public and consumable are not in their scope. In this paper, we present an architecture that can be used to provide a layer of consumable devices that can be applied in any domain and deal with those issues.

When considering systems that use sensors as a service, there is an interoperable system [7] that offers a web service description language interface designed for users to access sensors. However, its main focus is on low-power wireless sensor nodes and energy profile. The SOCRADES middleware [8] is a solution for business integration that offers a web service-oriented architecture that integrates different types of intelligent objects. It uses an enterprise business solution

system to provide the ability of users to construct services based on physical objects using the Web of Things [9] and using REST applications [10]. These approaches tie the design of the device to the web system restraining new devices to be added at runtime. In our proposed approach, devices have their own mechanism for dealing with the core system at runtime.

III. THE RESOURCE MANAGEMENT ARCHITECTURE

In this section, we present the Research Management Architecture (RMA), an architecture responsible for the virtualization of devices that act as IoT objects, exposing their sensors and actuators to be accessed and consumed by who might be interested in their functionalities. These resources' data are maintained at runtime in a model that can be publicly visualized and subscribed for several purposes. Exposing devices as a service that can be virtually consumed brings some advantages since users do not need to have their own devices, reducing eventual costs in creating new ones. In this way, users can provide a shareable resource component that can be exploited for several purposes. The proposed architecture (Figure 1) is composed of three different layers:

- 1) **Device:** in this layer, the devices have an embedded system enhanced with a processing cycle where (i) it connects and registers in the Cloud layer at the very first time; (ii) it gathers data from all its sensors to be sent to the cloud layer and; (iii) it receives from the Cloud layer actions that must be executed by the device's actuators.
- 2) **Cloud:** it is capable of maintaining updated information from devices hosted in environments and running over an IoT middleware. The Resource Management Component (RMC) manages (i) the registering process of devices; (ii) the sensors' data updating process, and; (iii) the actions that must be executed by devices. Besides, it also capable of exposing RESTful web services for several purposes.
- 3) **Client:** it is responsible for the visualization of devices and environments. In this paper, it is represented as a web solution. It implements a publish and subscribe mechanism for allowing developers and researchers in obtaining data from real sensors without possessing them and it offers an infrastructure for those who want to provide sensors as a service.

In the RMA, a device is an IoT object equipped with a micro-controller (hardware) where sensors and actuators are plugged in. The devices host an embedded system capable of connecting and registering to the Cloud layer for providing current data to be consumed by users. The embedded system dynamically registers itself in the RMC by sending all its functionalities (available resources, data and action commands) and the environment it is situated. Once connected, the device receives a confirmation and it starts to send data gathered from its sensors directly to the RMC, which keeps the most recent value available to be consumed or visualized in a web panel in the Visualization layer. Besides, the embedded system deals with the action commands coming from the RMC that need to be executed by the device's actuators.

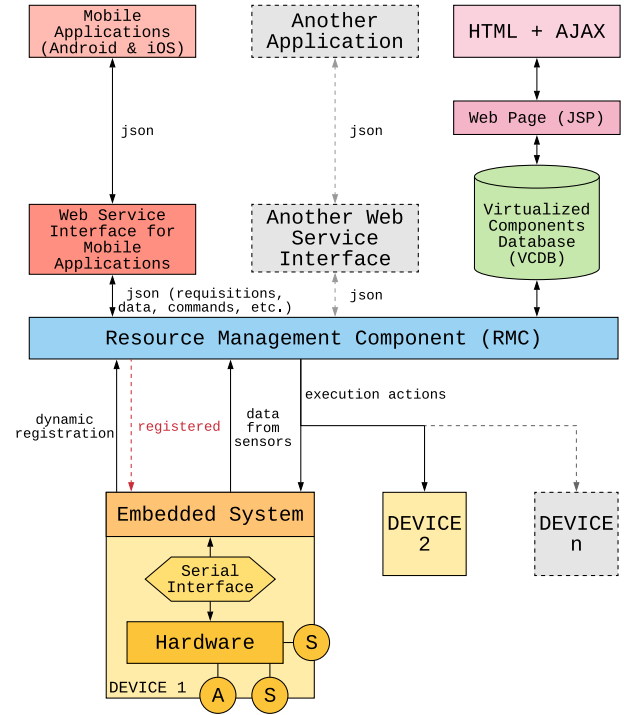


Fig. 1. The Resource Management Architecture's (RMA) components.

The RMC is the main component in the RMA and it maintains information about devices and environments to be consumed as a service by other layers. It has a mechanism for registering devices in given environments, storing their information in the Virtualized Components Database (VCDB). The VCDB keeps updated the data coming from devices by considering only the new values that have changed since the last data message was received. In addition, this layer exposes web services for providing different services such as visualization layers for mobile applications or allowing new technologies to interact and access the VCDB. Web services use a RESTful architecture for system interoperability and it uses Json files for exchanging information (data, requisitions, and execution commands) between the Cloud and Client layers.

The actions that need to be executed by devices are managed by the RMC. Any user that desire to perform an action in any available actuator must send an action command by accessing one of the provided services. The RMC redirects the action directly to the specific device. The technical information about hardware and actuators is transparent to the end user, that just need to know the available commands of the device he or she wants to interact. While one device is being used, the RMC blocks the specific actuator or the entire device for avoiding conflicts in the lower layer. The respective resource is unlocked after the execution or after a timeout boundary.

The RMC is a server-side solution running an IoT middleware instance where the devices must connect to. The middleware provides the connectivity and communicability necessary for devices to interact with the RMC layer and it

should guarantee the scalability of the system. These issues are not the focus of this work, however more technical details about the chosen middleware are provided later in Section IV.

For facilitating the environments managing, this layer provides a web page where all the environments can be accessed at real-time and users can create their own public or private environments. This web system shows all the public environments allowing users to select and see their available information and subscribe to have access to a specific and personalized group of information. In this work, an environment is defined by a logical representation of a physical space where several devices can co-exist sharing information in an IoT network.

IV. IMPLEMENTING THE RMA'S LAYERS

In this section, it is presented the implementation of the Device, Cloud and Client layers considering technological components and how these layers communicate with each other. For both Device and Cloud layers, we employ the ContextNet middleware [5], which is an IoT middleware for context reasoning and data sharing in a large scale environment. It is a context-providing service for stationary and mobile networks, which already addresses data communication issues such as fault tolerance, load balancing, node disconnect support (handover), and security. It uses the OMG DDS [11] protocol for handling messages between clients.

The Device layer uses a low coupled serial interface [12] for communicating with the micro-controllers that hosts the sensors and actuators. This interface isolates the high-level programming from the low-level using serial commands for activating users' pre-defined functions, which controls actuators or gathers data from sensors. The Client layer employs a web system capable of managing environments, showing their available resources, and it has a publish and subscribe mechanism for users interested in specific resources. All layers consider a model where these resources are part of devices situated in pre-existing environments in the architecture. For instance, the environments have to be manually registered by the user to facilitate their management and for security issues. This model is represented as a class diagram (Figure 2) shared between the three layers. The classes that can be found in our solution are described as follows:

- **Action:** the action that is executed at the low-level hardware in a device. Every command sent by the the Client and Cloud layers becomes an action in the Device layer.
- **Command:** it is the representation of commands available to be executed if the resource is an actuator. Sensors do not have commands because they are data providers.
- **Cycle:** it is the functioning cycle of the embedded system hosted in devices. It is responsible for synchronizing the device activities of sending data and executing actions.
- **Device:** it is the device representation used in the Client, Cloud and Device layers. It keeps the identification, name and description of a device and it is composed resources.
- **Embedded Client:** it is the ContextNet client instance responsible for receiving messages from the Cloud layer and other clients, and sending the data from sensors to

the Cloud layer. It also maps the components of the configuration file into its respective components.

- **Environment:** the virtual representation of environments in all layers. Each environment is composed of devices.
- **Main:** the main class that starts a device.
- **Resource:** it represents both sensors and actuators in all layers. The resources keep information about the serial port where the resource is connected, the available commands to be executed, and the availability of the resource.
- **Resource Management:** it is the main class of the cloud layer and it is a server instance of the ContextNet. It keeps a list of environments mapped and the devices registered for each one. It is responsible for the process of registering and updating devices and resources. It also exposes the web servers and the interfaces to the database.
- **Serial Communication:** the serial interface between the micro-controller and the system hosted at devices.

A. The Device Layer

The Device layer comprises devices running an embedded system with enough processing power interfacing sensors and actuators. There is a physical and logical architecture for clients where the first one uses hardware technologies and it is composed of a tiny mobile board with Bluetooth and WiFi connections (e.g. Raspberry Pi) connected to one or more micro-controllers using serial communication for accessing sensors and actuators. The logical architecture of the Device layer comprises both micro-controllers' programming and the embedded system. They are able of gathering the raw data from the sensors and then send it to the embedded system. After that, the embedded system sends the data to the Cloud layer or receive commands to be executed by actuators.

The micro-controller is programmed in a loop for verifying if there are messages coming from other layers. So, it accesses all the sensors and mounts a string to be sent by serial communication to the embedded system. Otherwise, the execution action is verified and if exists an equivalent programmed, an action is executed in the respective actuator.

Algorithm 1 Device's Processing Cycle

```

1: procedure CYCLE(configurationFile)
2:   mountDevice(configurationFile)
3:   intervalTime ← getIntervalTime()
4:   loop
5:     if isRegistered() then
6:       gatheredData ← dataFromSensors()
7:       sendToCloud(gatheredData)
8:       wait(intervalTime)
9:       actions[] ← getReceivedActions()
10:      executeNextActions(actions[])
11:     else
12:       registerDevice()

```

The embedded system controls the microcontroller and it uses a ContextNet client able to communicate with the Cloud layer. For this, there is a cycle (Algorithm 1) for synchronizing

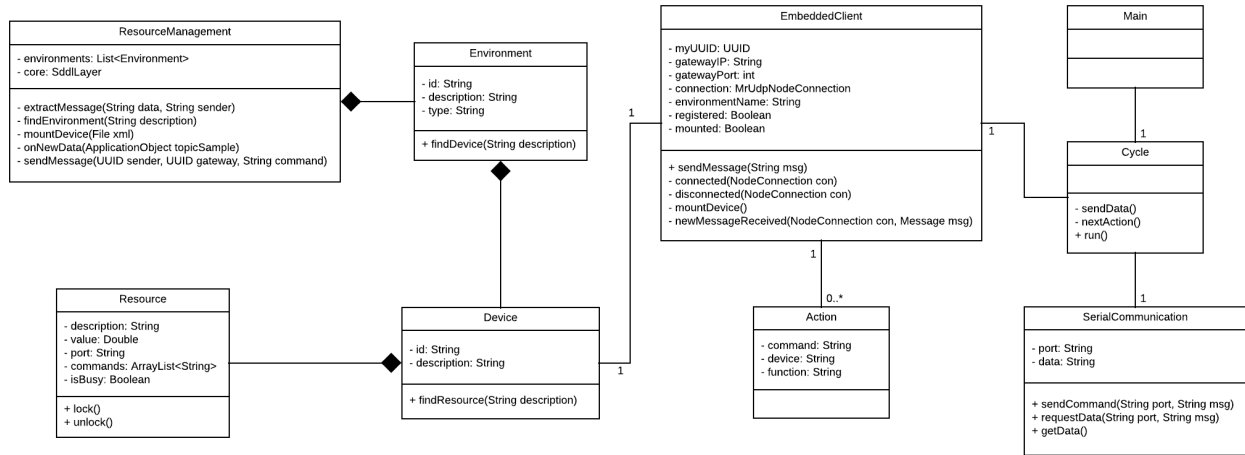


Fig. 2. The class diagram of the overall architecture comprising the Client, Cloud and Device layers.

the reception of data coming from sensors to be sent to the Cloud layer and the actions that need to be sent and executed in the micro-controller because both cannot execute at the same time for avoiding undesired conflicts. So, the actions received from the Cloud layer are put in a queue of actions to be executed one step after the data from sensors are collected through serial ports. It is also capable of performing a self-configuration and registration at the Cloud layer when it starts running and there is a cloud server available. For this, the device keeps an XML file describing all the available resources, commands, and the server configuration information. Once the file is correctly filled, the system performs everything automatically. The following information must be provided:

- (i) **Server:** the gateway of the server and the port must be provided in order to connect to the Cloud layer where the server instance of the ContextNet is installed. Besides, the time interval of sending data to the server must be set.
- (ii) **Environment:** for instance, it is necessary to inform in which environment the device is inserted into. For this, the identification number of the environment must be declared in the configuration file. The environment's identification is generated when users register an environment in the web system.
- (iii) **Resources:** all the resources available must be mapped in the configuration file. All the resources have the serial port where it is connected, its name, and a non-mandatory description. If the resource is an actuator, it also has the available execution commands.

The embedded system architecture is composed by the *Cycle* class, which instantiates an instance of ContextNet client (represented by the *EmbeddedClient* class) capable of exchanging message with other clients and the cloud server. Besides, the *EmbeddedClient* is responsible for the serial interface and it deals with a queue of actions received from the Cloud layer that has to be executed. The *SerialCommunication* and *Action* are the classes responsible for these behaviors.

B. The Cloud Layer

The cloud layer is a server instance of the ContextNet running a core system responsible for the virtualization of environments, devices and available resources. The devices register at the cloud informing their resources and environment where they are situated. This process starts when the core receives the file containing all the information of the device. Then, the system extracts the necessary information from the file and it registers the device at the system in the informed environment. Afterward, it sends back a message to the device authorizing the beginning of sending data from sensors.

Once the devices start sending data, the core system registers the new values and updates the old ones in VCDB (Figure 1). For every device, there are resources that can be sensors or actuators. In the case of sensors, if it is the first time that a new value of a resource is received, the system inserts this new value at the VCDB. Otherwise, if the value has changed since the last data reception, this new value is updated in VCDB. In the case of actuators, there is no data to be stored but it is kept the information about the availability of the resource. For example, the system informs if a certain actuator is being used or it is free for executing commands.

The core system deals with commands requisitions coming from the Client layer (a web system or by exposed web services) to be sent and executed at devices. The Client layer does not need to know technical details of hardware or even in which device the command will be executed. The commands are specific for a resource and the core system avoids duplicated resources and commands. Besides, it also redirects a received command to the respective device registered at the system by sending a message using ContextNet.

In the Cloud layer, there is also a locking process for avoiding conflicts in executing actions when two or more clients try to execute commands in the same actuator. For this, every time that a client needs to use a specific actuator, the core system locks this resource until the client informs that

is no longer using it, the action was performed, or a timeout boundary is reached. During this process, the locked resource is unavailable for all clients. It is important to remark that sensors are not part of the locking process because the nature of sensors is to provide data to be consumed. If the data is considered confidential, the environment can be set as private.

As stated before, web services can be exposed for extending the functionalities of the RMA in the Cloud layer. For creating RESTful web services it is used as an embedded servlet container and web server named Jetty. The available web services are a requisition service for agents applications for using resources with contextual planning; an execution service for activating or deactivating the actuators based on available commands; and a web service for providing data access to mobile applications.

The Cloud layer's model is composed of the *ResourceManagement* class where the ContextNet server is instantiated. Besides, it hosts a list of *Environments* with their *Devices*, *Resources*, and *Commands* classes.

C. The Client Layer

The idea behind the Client layer is to provide a layer capable of showing environments' resources in any kind of platform such as web pages, web services or mobile applications. Besides, it is responsible for providing some basic mechanisms that are not available in previous layers such as environments creation and, publish and subscriber mechanisms for example. The Client layer is represented as a web page for showing all the resources of environments and some basic functions for interacting with the resources. The user is able of creating environments to virtually host his devices and to expose them to be consumed by other users. Besides, the actuators have commands that users can activate by interacting with the Client layer's web system. The technologies employed are relational databases, Java web pages and Ajax.

Besides that, users can choose to follow some of the resources without the need to access the environments every time that he needs to get those values. The publish and subscriber mechanism allows users to access values always that a change is perceived by the core system in the Cloud layer. It allows the user to set basic rules such as defining a desired value to be announced when reached. All existing modules can co-exist without interfering in each other since all functions are managed by the Cloud layer. They all have to connect to the Cloud layer's database or use an exposed web service to access information.

V. EVALUATION

In this section, it is presented an initial case study evaluation in the assisted environment domain using the proposed RMA using a software engineering based approach. Case studies are employed because they are suitable for evaluation of software engineering methods involving development, operation, and its maintenance and artifacts [13].

The scenario will be held in a hypothetical Smart City where the government has access to a hospital where the

RMA is implanted. Some rooms in the hospital building have devices for controlling the temperature and luminosity (as sensors), and light lamps of the room (as actuators), and other devices for measuring some of the patients' information such as heartbeat frequency. Therefore, every room in the hospital endowed with devices is considered an environment in the RMA and its devices' resources are exposed as a service for the board of directors, government and everyone interested in them. It is important to remark that, even in this hypothetical scenario, there is no real personal contact of patients available.

Based on this, two devices were prepared for a room, named Room 403. Both devices use a Raspberry Pi Zero connected to an Arduino board. The first device is connected to a temperature and a luminosity sensor for the basic sensing of the room. The second one is a device with a light lamp connected to the Arduino working as an actuator and informing if the lamps are on or off. Besides, virtual devices were simulated in order to stress the RMC functioning. For this, the serial interface between the embedded system and the hardware were disabled, and several resources were simulated for each virtual device, which sends random data to the Cloud layer. So, one device for monitoring the heartbeat frequency of a patient and devices identical to the real ones above were simulated in each room. In general, 20 environments were prepared where the environment Room 403 has two real devices and one simulated, and the other 19 have three simulated resources.

The case study approach is divided into four steps: case study design, preparation for data collecting, the data collecting and data analysis. The following Table I shows the details and aim of the descriptive case study.

TABLE I
THE CASE STUDY DESIGN

Design	Description
Objective	A descriptive analysis of the behavior of the RMA functioning.
Case	The asynchronous process of transferring information from devices to the Cloud layer to be consumed by clients using web solutions.
Questions	Is the device connects correctly to the Cloud layer? Is the communication process between all layers works? Is the Client layer showing the correct data?
Method	Qualitative data analysis using negative case analysis and observation method.

Some tests were conducted trying to deny the research questions above. The preparation for the data collecting consisted in store both dynamic registration at the RMC and the answer that devices receive before starting sending data from sensors. Afterward, it is verified if these data arrives at all layers properly by analyzing the transferring process between hardware and device, device and cloud, and cloud and client. A string of data that comes out from the hardware is collected and compared if the data read arrived correctly at the embedded system. Between Devices and the Cloud layer, all devices should keep sending data to be stored at the VCDB in the RMC. Finally, between Cloud and Clients, these same

data should be read by clients when data update occurs.

The data collecting and analysis were performed in an arithmetic progression from 1 to 20 devices. Firstly, the server instance was running properly to verify the effectiveness of RMA and then it was disabled. Once there is no server instance available, they should not send data. Then, the data should be properly stored and read by Cloud and Client layers. Table II shows the resumed results from tests.

TABLE II
DATA COLLECTION AND ANALYSIS

Test	Description	Hit (%)
Hardware	Data is correctly transferred to the embedded system.	100
Connection with Server	Device registered at RMC and registered message received.	100
Connection without Server	Device registered at RMC and registered message received.	0
Data Updated and Stored	Data correctly stored at VCDB.	100
Data Read	Data correctly read by clients.	100

The communication between hardware and the embedded system is done using a serial interface, which guarantees no losses in the data transferring. None errors were observed in this process. As expected, when the server instance is disabled, it is observed that devices try to connect to the Cloud layer but there is no response from the server and no data is sent from any device. Otherwise, the device is registered and receives a confirmation to start sending data to the Cloud. All devices work properly considering an available server instance.

The most recent information available coming from devices is updated in the VCDB. Considering that the VCDB is implemented as a relational database, there are no big deals in this process not even in the visualization of the devices by the Client layer. This case study focused on observing the communication and the correctness of data flowing through the architecture. More experiments focusing on performance and a proper formalization were left for future efforts.

VI. CONCLUSION

In this work, it was presented a low-coupled three-layer architecture for exposing devices as a service to be consumed by clients. The devices are able of connecting to an IoT server, registering their resources (sensors and actuators) in a core system, which turns all the public data available that can be accessed by clients using web services and a web platform. A case study was proposed and evaluated integrating the Device, Cloud and Client layers for monitoring an environment.

RMA architecture employs different technologies in its layers. Devices are autonomous and uncoupled from the Cloud layer because they are built using hardware platforms enhanced with wi-fi connections and it uses a serial interface for communicating with micro-controllers. These technologies provide the necessary autonomy, heterogeneity of hardware employed, and communicability to the Cloud layer. The ContextNet middleware provides client and server instances and it

is used in the architecture because of the middleware guarantees connectivity, communicability, reliability, and scalability, in addition to using an industrial market standard protocol.

The Client layer offers web solutions for managing environments and for the visualization of their respective resources. Besides, it is possible to subscribe specifically to resources that one might be interested in. Moreover, the RMA aims to provide an architecture for exposing devices as a service to be consumed by persons that do not have access to these kinds of resources either for the cost or complexity of creating from scratch an architecture for that purpose.

Nowadays, the designer of the device should program how data is mounted and captured by the micro-controller and sent to the embedded system. As future works, it is important to create an automatized plug-and-play way of configuring the device in low-level. Besides, as micro-controllers are connected to the serial port of the tiny computer of the device, a similar process for the identification of serial ports by the embedded system is also interesting. The environment has to be set manually at the device's configuration file for security and control reasons, nevertheless, it is possible to identify and register autonomously the environment based on access points.

REFERENCES

- [1] E. Santos, P. H. V. Penna, I. M. Coelho, H. D. Soares, L. S. Ochi, and L. Simonetti, "Logistics sla optimization service for transportation in smart cities," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2018.
- [2] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a Service Model for Smart Cities Supported by Internet of Things," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 3, pp. 294–307, 2014.
- [3] M. Weiser, "Some computer science issues in ubiquitous computing," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, pp. 12–, July 1999.
- [4] C. Pantoja, H. Soares, J. Viterbo, and A. Seghrouchni, "An architecture for the development of ambient intelligence systems managed by embedded agents," in *The 30th International Conference on Software Engineering & Knowledge Engineering*, (San Francisco), 2018.
- [5] M. Endler and F. S. e Silva, "Past, present and future of the contextnet iomt middleware," *Open Journal of Internet Of Things (OJIOT)*, vol. 4, no. 1, pp. 7–23, 2018.
- [6] P. Neirotti, A. De Marco, A. C. Cagliano, G. Mangano, and F. Scorrano, "Current trends in smart city initiatives: Some stylised facts," *Cities*, vol. 38, pp. 25–36, 2014.
- [7] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pp. 253–266, ACM, 2008.
- [8] L. M. S. De Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio, "Socrates: A web service based shop floor integration infrastructure," in *The internet of things*, pp. 50–67, Springer, 2008.
- [9] D. Guinard and V. Trifa, "Towards the web of things: Web mashups for embedded devices," in *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web*, in *proceedings of International World Wide Web Conferences, Madrid, Spain*, vol. 15, 2009.
- [10] B. Ostermaier, F. Schlup, and K. Römer, "Webplug: A framework for the web of things," in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops*, pp. 690–695, March 2010.
- [11] G. Pardo-Castellote, "Omg data-distribution service: Architectural overview," in *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pp. 200–206, IEEE, 2003.
- [12] N. M. Lazarin and C. E. Pantoja, "A robotic-agent platform for embedding software agents using raspberry pi and arduino boards," *9th Software Agents, Environments and Applications School*, 2015.
- [13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

SPRO: SECURITY PROCESS FRAMEWORK

Henrique Michel Persch, Lisandra M. Fontoura
Departamento de Computação Aplicada (DCOM)
Universidade Federal de Santa Maria (UFSM)
Santa Maria, Brasil
{hpersch, lisandra}@gmail.com

Adriano Brum Fontoura
Instituto Federal Farroupilha
Santa Maria, RS, Brasil
fontoura.ab@gmail.com

Abstract— Secure software implies that the process used in its development includes activities to insert, monitor and ensure security from the early stages of the software process. This article proposes the Security Process Framework that aims to facilitate the task of creating secure processes through the reuse of process components developed from security patterns. The components are recovered and prioritized from a repository through multicriteria techniques that consider security requirements and characteristics related to the project context. Software process lines are used to organize the selected components and to assemble the secure software process. Furthermore, a tool called SPro System was developed to support the use of the framework. Case studies were used to verify the applicability of the proposal, which showed that the framework and SPro System facilitated the tailoring and decreased the time spent in the definition of security processes.

Keywords - security patterns, security process, software process lines, process tailoring

I. INTRODUCTION

Nowadays, software are becoming increasingly complex, requiring high levels of security to prevent the occurrence of failures and data loss. Several authors propose that methodologies and processes for software development should be concerned with the security from the early stages of projects life cycle [8][9].

Nevertheless, to create software processes aimed at a secure software development is a long process, time consuming and that requires many resources. In addition, the processes must be suited to the needs of each project, which requires defining activities and security practices appropriate to the context and that includes the security requirements defined for the project [3][4][18]. This task is not trivial and it requires experts in security and in software processes, which are not always available in the team.

In the literature, numerous security standards and models are presented which describe best practices and security requirements, such as: ISO/IEC 27001 [1], ISO/IEC 27002 [2] and ISO/IEC 21827 [10]. However, these standards, as any other standard, do not detail the processes that need to be followed to implement security in its projects.

To facilitate the task of setting secure software processes, this paper proposes a framework consisting of: (1) a repository of process components defined from the security patterns; (2) techniques for prioritization and selection of these components from context information; (3) process lines that organize

process components contributing to their reuse and (4) a tool to support the use of the framework.

The components stored in the repository were defined from security patterns catalogs [15], security standards [1], [10] and scientific literature. Security patterns provide solutions already established to recurring security problems and serve as a reference for organizations that seek to meet security requirements [15]. Process components facilitate the development of new processes through the reuse of predefined components and enable the assessment and improvement of these through the use in different software projects. The components are associated with security requirements through rules and are prioritized from multicriteria techniques that consider the context of the project. The prioritized components are arranged using software process lines.

This article is organized as follows: in Section 2, related studies are discussed; in Section 3, concepts relevant to the understanding of the study are introduced; in Section 4, the proposed framework is described showing how the components of the processes and the prioritization methods were defined; in Section 5, the support tool is explained; and in Section 6, a case study is presented. Finally, in Section 7, the final considerations of the study are described.

II. RELATED WORK

In [11], a methodology for adapting software processes is presented, based on security requirements recommended by the security practices of the ISO/IEC 21827. The processes are compiled from components of the Rational Unified Process (RUP), Extreme Programming (XP) and on security patterns proposed in the literature. This study was developed in the same research group and the main differences is that it does not consider the context of the project, the use of prioritization techniques and software process lines.

Mellado, Medina and Piattini [6] propose to incorporate security requirements from the early stages of development using software product lines. Thus, the authors seek to facilitate the compliance with security standards and to manage potential variabilities that may happen among security requirements. The authors propose the use of security standards (ISO/IEC 27001 e ISO/IEC 15408) to manage security requirements. The focus of the work is on the product and our work focuses on the process.

In [7], the authors propose an extension of the Scrum software development framework with features focused on creating secure software.

Hamid and Weber [14] propose a model-driven engineering (MDE) methodological approach associated with a pattern-

based approach to support the development of secure software systems.

However, this study differs from the others due to the development of a secure processes tailoring framework using process components associated with the process area from ISO/IEC 21827. The process components are prioritized from multi-criteria decision techniques that consider context information. In addition, software process lines are used which organize the recovered components to form the process and a support tool is provided.

III. BACKGROUND

Security standards and models have key goals and practices so that organizations can define the expected level of security in their processes. In the literature, various standards of information security are described, among which we can highlight: ISO/IEC 27001, ISO/IEC 15408 and ISO/IEC 27002. ISO/IEC 27001 [1] describes a process for information security management in an organization that structures activities through a continuous improvement cycle. On the other hand, ISO/IEC 15408 provides a common set of requirements for security functions in products and systems and for assurance of measures applied to them during a security assessment [12]. The model SSE-CMM - Systems Security Engineering Capability Maturity Model, published as ISO/IEC 21827: 2008 [10], describes a set of process areas (PAs) that are required in a security engineering process. ISO/IEC 21827 proposes 22 process areas (PAs) that are organized into two groups: Security Base Practices and Project and Organizational Base Practices [10]. For each PA, it is presented a list of BPs (Base Practices) that assist in meeting the goals of the process areas (PA).

Security standards and models describe ways that are intended to assist in the development of security processes, providing practices to be implemented or guidelines to be followed by organizations. In this study, we chose to use the ISO/IEC 21827 [10] because this model defines a set of best practices that guide the organization in the implementation of effective security processes. In addition, this model is widely known and widespread.

Nevertheless, the ISO/IEC 21827, as well as other standards, does not detail the activities to be implemented by organizations. Therefore, it was decided to seek solutions in the literature used by other organizations that have been successful. Many of these solutions in the security area are documented as security patterns [15]. Even though many security patterns and techniques to use them are being proposed, it is complex to adapt and integrate them in every stage of the software development or in specific contexts.

In this study, the patterns are described as process components and associated with a specific use context. However, describe components and store them in a repository does not guarantee the proper selection and integration of these components to create a consistent software process. To solve this issue, the process components are selected through multi-criteria techniques and organized using software process lines (SPrL). SPrL are intended to represent the dynamic aspect of

the processes and a metamodel is used to describe the structural aspect of the components.

Software process lines emerged from software product lines and are aimed at the development of consistent processes, enabling the reuse of previously defined components. SPrL is a form of tailoring of processes that has the following objectives: i) increase the quality and adequacy of the processes; ii) representation of variabilities and similarities among processes to maximize reuse; and iii) reduce the risks of an inadequate tailoring of process [16].

The multi-criteria techniques help the decision making from the analysis of preferred criteria necessary for understanding the reality of the analyzed problem and the choice of the alternative that will allow the best decision to be made. These are examples of multi-criteria methods: AHP (Analytic Hierarchy Process) [19], TODIM (an acronym in Portuguese for Iterative and Multi-criteria Decision Making) [13], and TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution) [5].

The theoretical foundation of the AHP is based on the decomposition and synthesis of the relations among the criteria until a prioritization of their indicators is reached, approaching a better response of single measurement of performance [19]. The TODIM is based on the Prospect Theory [13]. On the other hand, the TOPSIS is based on the calculation of the Euclidean distance between the most beneficial alternative and the costlier alternatives [5].

In our study, the prioritization methods are used to select the most appropriate process components for the project in question, using criteria that define the context. Several authors propose that the context should be considered for definition of software processes, such as: Boehm and Turner [4], Cockburn [3] and Kruchten [18]. In this study, we chose to use the context described by the Octopus Model, as proposed by Kruchten [18], which uses the following contextual factors: size, stable architecture, business model, team distribution, rate of change, system age, criticality and governance.

IV. SECURITY PROCESS FRAMEWORK

This section describes the Security Process Framework through detailing the process components repository and metamodel, the rules of components association to security requirements and the techniques used for the prioritization of the components.

A. Repository of Security Processes

The process components were defined from classes described in the metamodel (Figure 1), elaborated according to Software and Systems Process Engineering Metamodel Specification (SPeM) [17].

A process component is an aggregation of tasks that are performed by roles which have artifacts as input and output and are associated with a discipline. Furthermore, project processes are composed of process components and are associated with projects that use the process.

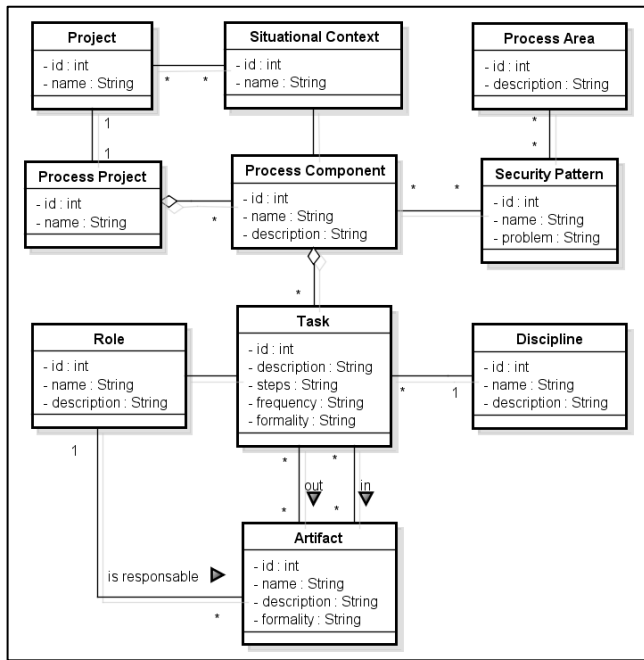


Figure 1. Security Process Metamodel

Process components and projects are associated with information of the situational context. Process components are associated with security patterns from which they were defined. Security patterns [15] are associated with process areas that they meet, extracted from the ISO/IEC 21827 [10].

To illustrate how the classes described in the metamodel are used to define the components, Figure 2 shows the process component Asset Valuation, described from the pattern of the same name [15]. This pattern aims at helping the organization to determine the overall importance of the assets under its control and ownership.

B. Security Components Association to PAs

The defined process components are associated with the goals of the 11 PAs - security base practices, found in the standard ISO/IEC 21827 [10]. For example, the component Asset Valuation (Figure 2) was associated with the process area PA 03 - Security Risks Assessment.

After the analysis of several catalogs of security patterns, each security pattern was described as a process component and associated with one or more process areas that it covers. These components were placed in a repository and are available for use in the development of processes that aim to meet security requirements. Thenceforth, it was possible to build a repository of software process components based on the ISO/IEC 21827 [10], with 50 process components, which have about 70 artifacts and 60 tasks.

C. Determination of the Prioritization Methods

The multi-criteria methods are used to select process components that have a context of use similar to the project context. These contexts are described using the attributes proposed by the Octopus Model.

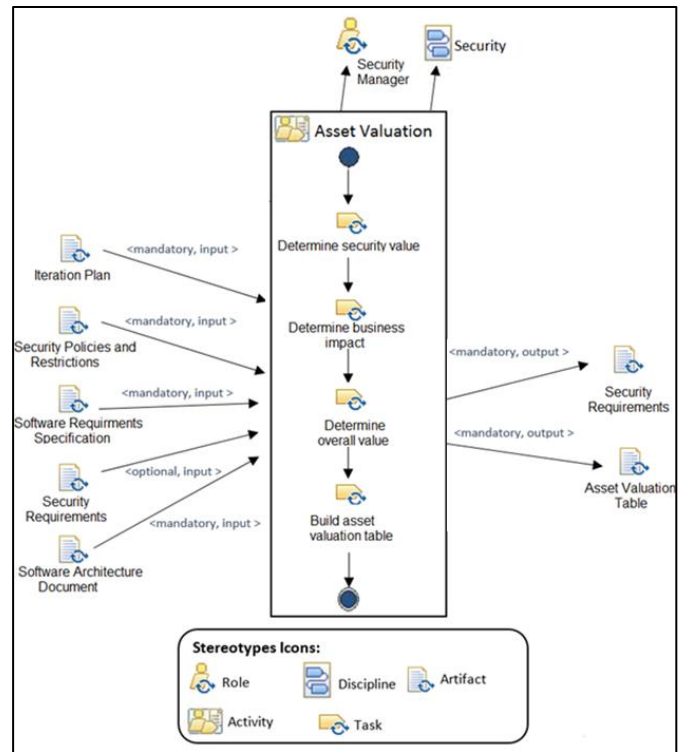


Figure 2. Details of the process component Asset Valuation

In Figure 3, the process area PA08 - Assess Security Risk is shown associated with four components of different processes that aim at satisfying it. The contexts of the component and of the project are assessed and compared according to the calculations proposed by each prioritization method to assist in choosing the best components.

V. SPRO SYSTEM

In this section, a tool that supports the use of the framework for tailoring of processes, using software process lines, called SPro System is described. Process components defined from RUP and XP were inserted into the repository. Process lines have been defined to organize the components and architectures to represent the variants.

The processes tailoring are performed through four stages, which are: i) definition of the project context; ii) selection of the tailoring requirements and of the processes architecture; iii) prioritization of activities; and iv) creation of the tailored process. To illustrate the stages, the implementation of an example of use is described.

The situational context of the project is defined in order to select the most appropriate process components for the project. This example refers to a new development in a well-known domain that involves only loss of money and it is being developed by a local team of 25 people, the values were defined as follows: size = medium, change of rate = less than 10, type of architecture = new, age of the system = new development, business model = commercial, criticality = loss of money, distribution of the team = local and governance = simple rules.

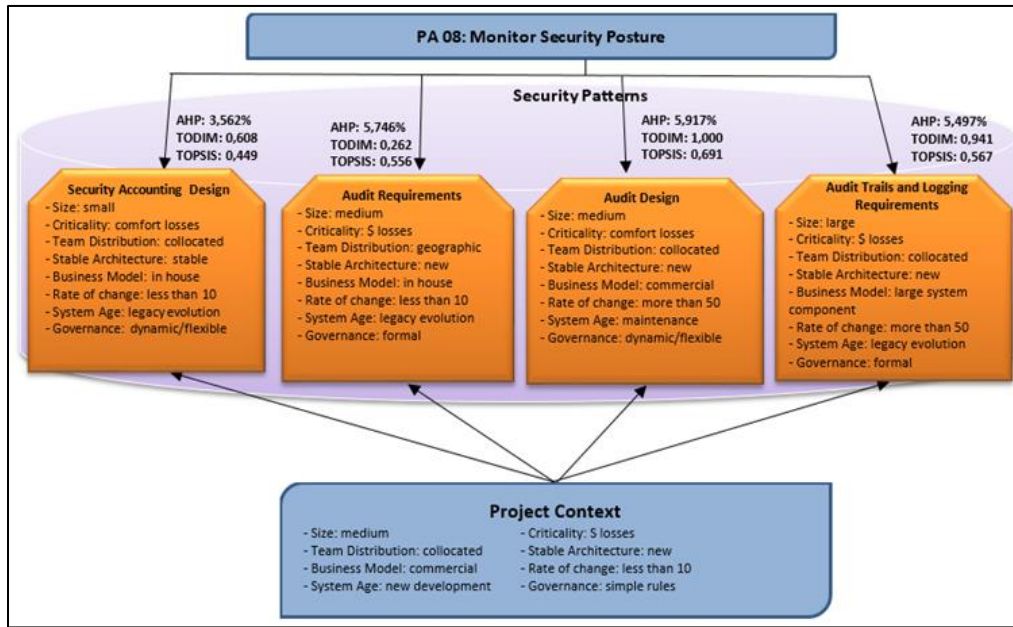


Figure 3. Selection of the Process Patterns

In the second stage, the process engineer selects the tailoring criteria and the architecture. For the tailoring criteria, the PAs 03, 05, 07 08, 10 were selected as showed in Figure 4 (A). In the tool, different tailoring criteria can be used, such as risks, quality, however, this article is limited to describe on tailoring criteria based on the ISO/IEC 21827.

For each selected tailoring criterion, the SPro System searches in the repository the process components that were previously associated with the tailoring criteria through rules, that is, the components that aim at meeting the tailoring criterion. For these components, the similarity between the context of the project and of the component is calculated. The similarity values generated by the application of multi-criteria techniques help process engineers' decision-making, however, the selection is their responsibility. The multi-criteria techniques are an important source of information because they are based on different methodologies of analysis, prioritizing the most relevant activities for the development in particular. Figure 4 (B) shows the results of the prioritization of the components. From the defined process architecture and the recovered process components, prioritized and selected, the specific process for the project was created (Figure 4 C).

VI. VALIDATION USING CASE STUDY

For the validation, a case study was carried out, which was applied to a class of the Master's course in Computer Science from the Federal University of Santa Maria in the discipline of Software Process Improvement. The following materials were used for the experiment:

- A set of 11 cards with the name of the PAs proposed by the ISO/IEC 21827, which are the tailoring criteria to be selected;
- A set of 45 cards representing the process components, elaborated from the security patterns, contextualized

according to the attributes proposed by the Octopus Model;

- Description of two scenarios of development of a software;
- Description with examples of associations of PAs with process components elaborated from the literature;
- Sheet of paper to be filled in by the group with the PAs and the process components selected for the project described in the scenario.

Initially, the students were divided into 2 groups with 4 members each. Each group received the support material described above. The group's objective was to develop a software development process from the scenario. To this end, they needed to identify which PAs should be selected in the described project and, then, select the appropriate process components to the PAs and to the context of the project. In the first stage, the experiment was performed without SPro System and in the second stage the SPro System was used. In each stage, a different scenario was used.

A. Observed Results

The most striking factor was the decrease in time to elaborate the process. The time spent in the first stage was about 1 hour and 15 minutes. In the second stage, the average time was 25 minutes, thus the SPro System helped the team to reduce the time by approximately 60%.

Regarding the context of the project, the two groups correctly extracted the characteristics of each project from the provided scenario. According to the analysis performed in the process architectures developed in the first stage, it was possible to verify that group 1 was able to determine the exact process areas that encompassed the scenario. In contrast, group 2 included two extra process areas, which were not required by the scenario.

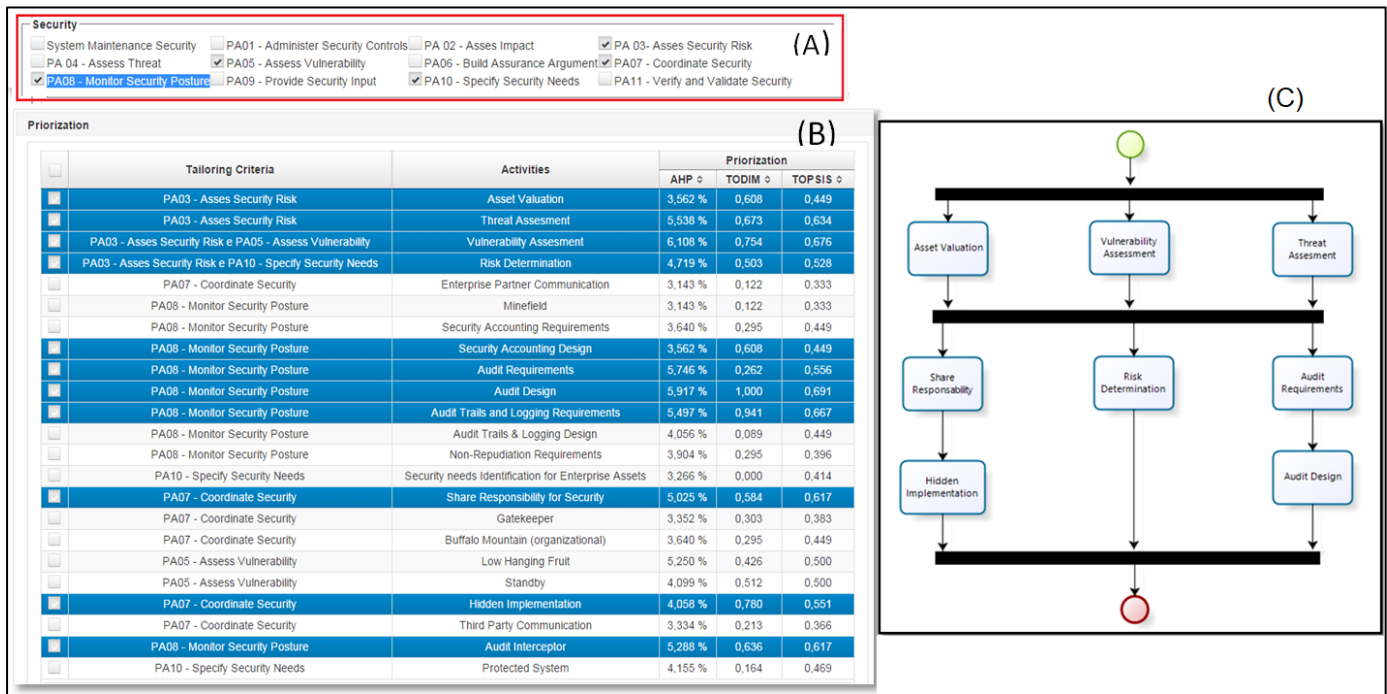


Figure 4. SPro System

Regarding the process components selected for each tailoring criterion, it was possible to verify that group 1 selected 10 activities for 5 tailoring criteria. Thus, it was verified that this group correctly compared the contexts of the activities and the project, besides an analysis of the description of the activity. Group 2 has determined 19 activities for 5 tailoring criteria, however, it was found that this group ignored the context and the activity characteristics, giving greater attention to the description of the activity. In this stage, both groups were able to determine the activities and the tailoring criteria considered as essential in the presented scenarios.

In the second stage, the activity was developed according to the first step, however using the SPro System and with another scenario. It is possible to state that the groups obtained more satisfactory and significant results when using the tool because they selected the activities and the tailoring criteria considered as essential in each presented scenarios

B. Assessment Questionnaire

To complement the analysis, a questionnaire was applied so that the students could assess the framework and the SPro System. The questionnaire responses are showed in the Figure 5.

All participants agreed that the use of the tool facilitated the processes tailoring aiming at meeting the security requirements (Q.4). Furthermore, they fully agreed on the importance of maintaining a repository with adapted process components to facilitate new adaptations (Q.8). Regarding the ease of understanding and the translation of the security patterns for the activities so that they can be added to a software process, there was a partial agreement (Q.2).

Only one participant partially disagreed that the provided patterns were sufficient to meet the tailoring criteria (Q3), wherein four answers partially agreed and the other 3 totally agreed. In this sense, it can be highlighted that the groups considered that the use of security patterns may be a viable solution to solve the security problems in the project, however, it may not be the only solution that should be applied.

The participants recognized that the pre-defined process components associated with the tailoring criteria facilitated the development of the process lines (Q.5). All participants agreed that the SPro System tool assists in the processes tailoring considering security requirements (Q.6).

Nonetheless, when asked if the tool expresses the stages required to define processes, there were different responses, wherein four participants fully agreed, three partially agreed and one neither agreed nor disagreed (Q.7). From these responses, there was the need to improve the stages to be used in the tool for the processes tailoring. In addition to this improvement, in the descriptive responses, suggestions regarding the SPro System were given.

Moreover, in the space for comments, the participants reported the difficulty in detecting security patterns and activities that meet the tailoring criteria.

Regarding the prioritization methods, it was possible to verify that the groups thoroughly analyzed the numbers obtained and brought in the calculations made by the AHP, TODIM and TOPSIS methods. According to the participants, the methods were an excellent source of information and guided the groups in the decision-making regarding which process components they should select.

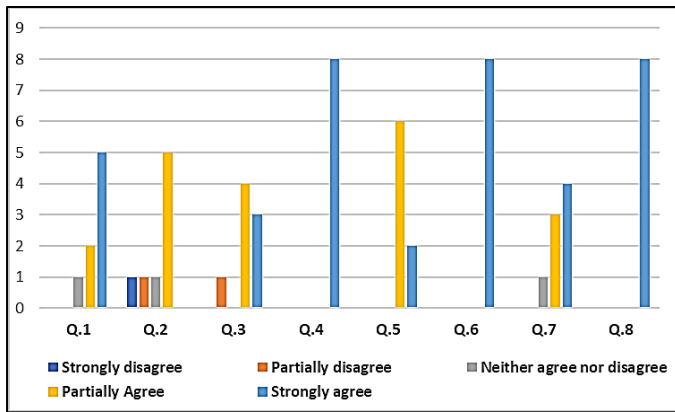


Figure 5. Responses of the applied questionnaires

From the development of this case study, it was possible to evaluate the use of the SPro System tool and to apply the concepts presented in the elaborated framework.

VII. FINAL CONSIDERATIONS

This paper presented a framework proposal that aims at selecting process components that can be used for the tailoring of security processes. The process components were developed and derived from security patterns and were proven efficient in the development and tailoring using software process lines. Furthermore, with the use of multi-criteria techniques, the framework assists the process engineer to choose the most relevant components for the development process, considering the contexts of project and of the components. With the support of the SPro System tool, the use of the proposed framework was demonstrated.

The validation showed that the framework and the SPro System facilitated the process tailoring and decreased the time spent in the definition of security processes. In addition, the students, even without security expertise, were able to select activities appropriate to the scenario, properly creating the process according to the problem they received.

As future studies, we suggest the semi-automatic selection of process components and the expansion of the repository comprising other security standards.

VIII. REFERENCES

- [1] —. 2013. ISO/IEC 27001:2013 - Information technology - Security techniques - Information security management systems - Requirements. 2013.
- [2] —. 2013. ISO/IEC 27002:2013. Information technology -- Security techniques -- Code of practice for information security controls. 2013.
- [3] A. Cockburn. 2004. Crystal Clear: A Human-Powered Methodology for Small Teams. Addison-Wesley, Boston, 2004.
- [4] B. Boehm, R. Turner. 2009. Balancing Agility and Discipline. Pearson Education, Inc, Boston, 2009.
- [5] C. L. Hwang, K. Yoon. Multiple attribute decision making; methods and applications, In: Proc. Lecture Series in Economics and Mathematical Systems, Berlin, Germany, Springer-Verlag, 1981.
- [6] D. Mellado, E. Fernandez-Medina, M. Piattini. Security Requirements Variability for Software Product Lines. In: Third International Conference on Availability, Reliability and Security, 2008.
- [7] C. Pohl, H. Hof. Secure Scrum: Development of Secure Software with Scrum. International Conference on Emerging Security Information, Systems and Technologies (SECURWARE) 2015, Venice, Italy, 2015.
- [8] E. Amoroso, "Recent Progress in Software Security," in IEEE Software, vol. 35, no. 2, pp. 11-13, March/April 2018.
- [9] S. HUSSAIN, G. RASOOL, M. ATEF, A. K. SHAHID. A Review of Approaches to Model Security into Software Systems, 2013. Journal of Basic and Applied Scientific Research, ISSN 2090-4304. Lahore, Pakistan.
- [10] ISO/IEC 21827. Associação Brasileira de Normas Técnicas. NBR ISO/IEC 21827:2008 Information technology. Security techniques. Systems Security Engineering. Capability Maturity Model (SSE-CMM). Switzerland, 2008.
- [11] R. Wagner, L. M. Fontoura, A. B. Fontoura. Using Security Patterns to Tailor Software Process. In: Proc. International Conference on Software Engineering and Knowledge Engineering (SEKE), 2011. pp. 672-677.
- [12] ISO/IEC15408. ISO/IEC15408 - Information technology — Security techniques — Evaluation criteria for IT security. 2005.
- [13] L. F. Gomes, M. P. Monica, L. A. D. Rangel. An application of the TODIM method to the multicriteria rental evaluation of residential properties. European Journal of Operational Research, 193, 204–211, 2009
- [14] B. Hamid, D. Weber. Engineering secure systems: models, patterns and empirical validation. Computer & Security. 77, 315–348 (2018).
- [15] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, P. Sommerlad. Security Patterns - Integrating Security and Systems Engineering. John Wiley & Sons, 2013.
- [16] O. Jaufman, J. Münch. Acquisition of a project-specific process. Springer, Berlin, Heidelberg, 2005. pp. 328-342.
- [17] OMG. 2008. Software & Systems Process Engineering Meta-Model Specification, Version 2.0. Software Engineering Institute, Carnegie, Pittsburgh, 2008.
- [18] P. Kruchten. Contextualizing Agile Software Development. Vancouver, BC, Canada Proc. European System, Software & Service Process Improvement & Innovation (EuroSPI) 2010, 2010.
- [19] T. L. Saaty, Theory and Applications of the Analytic Network Process: Decision Making with Benefits, Opportunities, Costs, and Risks. RWS Publications, Pittsburgh, 2005.

Detecting Security Vulnerabilities using Clone Detection and Community Knowledge

Fabien Patrick Viertel¹, Wasja Brunotte¹, Daniel Strüder², Kurt Schneider¹

¹ Software Engineering Group, Leibniz University Hannover, Hannover, Germany

² Software Engineering Division, Chalmers | University of Gothenburg, Gothenburg, Sweden
{fabien.viertel, wasja.brunotte, kurt.schneider}@inf.uni-hannover.de, danstru@chalmers.se

Abstract— Faced with the severe financial and reputation implications associated with data breaches, enterprises now recognize security as a top concern for software analysis tools. While software engineers are typically not equipped with the required expertise to identify vulnerabilities in code, community knowledge in the form of publicly available vulnerability databases could come to their rescue. For example, the Common Vulnerabilities and Exposures Database (CVE) contains data about already reported weaknesses. However, the support with available examples in these databases is scarce. CVE entries usually do not contain example code for a vulnerability, its exploit or patch. They just link to reports or repositories that provide this information. Manually searching these sources for relevant information is time-consuming and error-prone. In this paper, we propose a vulnerability detection approach based on community knowledge and clone detection. The key idea is to harness available example source code of software weaknesses, from a large-scale vulnerability database, which are matched to code fragments using clone detection. We leverage a clone detection technique from the literature, which we adapted to make it applicable to vulnerability databases. In an evaluation based on 20 reports and affected projects, our approach showed good precision and recall.

Security; Code Clones; Information Systems

I. INTRODUCTION

In today's interconnected world, security is one of the most important challenges for companies and institutions [16]. Vulnerabilities in a software system allow hackers to intrude and maliciously alter its behavior. The impact can range from minor, such as bypassing the copyright of a movie, to major, such as the malicious intrusion into a control system of a nuclear reactor [4]. An example for the latter case is the Stuxnet worm, which used a weakness in a vendor driver library to infect 100.000 systems worldwide and inflict physical damage. Consequently, organizations begin assigning a higher priority to security as a quality attribute in software development.

A key challenge is to check the complete source code for
DOI reference number: 10.18293/SEKE2019-183

vulnerabilities to avoid the associated exploits. Since software developers are not equipped with the required security expertise, ideally, security experts should review the whole source code of a project. However, in the face of realistic projects that often include hundreds of thousands of lines of code, a manual check of the project by security experts is infeasible. To the rescue may come available community knowledge from vulnerability databases, such as the Common Vulnerabilities and Exposures (CVE) [18]. The CVE provides detailed knowledge about a large number of reported security flaws, including their impact. Developers may want to leverage this knowledge by detecting instances of the flaws in their projects. Unfortunately, the associated manual process is time-consuming and error-prone: Developers have to use a search engine in order to find relevant entries based on the names of the used libraries. Then they must manually scan the source code to uncover problematic uses of the affected libraries. Even worse, support with available examples in these databases is scarce. CVE entries usually do not contain an example exploit or patch, but just a link to a report or to a repository that provides additional information on proof of concepts, patches, and exploits.

In this paper, we address the following research question: *How can we harness available community knowledge to facilitate the detection of security vulnerabilities in software code?* We present an approach that uses *code clone detection* to detect instances of known vulnerabilities in source code. Clone detection aims to locate exact or similar code snippets, called *clones*, in or between software systems [21].

Our approach, illustrated in Fig.1, involves on a *security code repository* that contains security-relevant code snippets. Each snippet instantiates a known vulnerability.

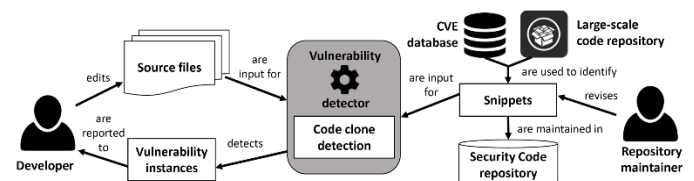


Figure 1. Approach Overview

The security code repository is created in a semi-automated process using automated searches over the CVE database and a large-scale code repository such as GitHub [6], and manual

refinement by developers. We detect duplicates of entries by using clone detection. To this end we have adopted an existing technique called SourcererCC [19], which fulfills two main prerequisites of our approach: It is efficient, as it scales to huge code bases with 100K LoC, and language-independent, as it supports arbitrary programming languages.

Our contributions are as follows:

- A *vulnerability detection technique* that uses and adapts an existing clone detection technique in order to detect security vulnerabilities, based on the given security code repository (Sec. III).
- A process for creating a *security code repository* with code snippets that instantiate known vulnerabilities, together with an initial version of such a repository (Sec. IV).
- An evaluation, in which our approach was able to detect the considered vulnerabilities with high precision and recall (Sec. V).

Clone detection has been used during vulnerability detection before. However, previous approaches were mostly limited to a particular programming language and suffered from scalability issues. We discuss related work in Sec. VI.

II. BACKGROUND

We recall a common taxonomy of code clones and its implications for security.

Clone types. The common taxonomy of code clones [9] distinguishes four clone types, based on the degree of similarity: *Type-1 clones* are code fragments that are accurate copies of each other, excluding whitespaces, blank lines, and comments. *Type-2 clones* are structurally identical code fragments that may differ in the names of variables, literals and functions. *Type-3* or *near-miss clones* are syntactically similar code fragments that, opposed to Type-1 and Type-2 clones, may include changes like added or removed statements. *Type-4 clones* are code fragments with a different syntax, but similar semantics. The example in Fig. 2 shows a code fragment CF_0 together with each clone type.

Security considerations. The clone types have different implications for security vulnerabilities. A vulnerability in a code fragment most likely also affects Type-1 clones of that fragment, since in most programming languages white spaces, blank lines and comments do not change the behavior. Neither does the change of variable names in Type-2 clones. However, the change of literal and method names can have an impact: a vulnerability may only occur when a specific method is called or when specific literals are used. Type-3 are particularly challenging for our approach, since an added line may render an insecure fragment secure, and vice versa. For example, consider the infamous buffer overflow weakness, where the problem is that the buffer size is not checked before writing or reading of it. A range check before accessing the buffer would fix this error, but the resulting code fragment is still a Type 3 clone. Type-4 clones regard the semantics of code snippets. The security impact of these clones depends on the chosen semantic representation. The typical means for checking semantic equivalence (such as pre- and post-conditions) are orthogonal to

contained security vulnerabilities. Therefore, we do not consider Type 4 clones in our approach.

Initial Code Fragment CF_0	CF_1 – Type-1 Clone	CF_4 – Type-4 Clone
<pre>for(i = 0; i < 10; i++) { // foo 2 if (i % 2 == 0) a = b + i; else // foo 1 a = b - i; }</pre>	<pre>for(i = 0; i < 10; i++) { if (i % 2 == 0) a = b + i; //cmt 1 else b = b - i; //cmt 2 }</pre>	<pre>while(i < 10) { // a comment a = (i % 2 == 0) ? b + i : b - i; i++; }</pre>
CF_2 – Type-2 Clone	CF_3 – Type-3 Clone	
<pre>for(j = 0; j < 10; j++) { if (j % 2 == 0) y = x + j; //cmt 1 else y = x - j; //cmt 2 }</pre>	<pre>for(i = 0; i < 10; i++) { // new statement a = 10 * b; if (i % 2 == 0) a = b + i; //cmt 1 else a = b - i; //cmt 2 }</pre>	

Figure 2. Clone-Types 1 to Type 4

III. VULNERABILITY DETECTION

Our vulnerability detection approach involves four steps: *Pre-Processing*, *Code Processing*, *Clone Detection* and *Results*. The Pre- and Code Processing consists of the substeps Parsing & Tokenizing and Indexing. Only the Pre-Processing also contains the step of the CVE Data linking to enrich the code snippets with meta-information out of the CVE.

The input for the Pre-Processing step are the vulnerable code snippets of the security code repository including their assigned CVE metadata. During parsing, we identify contained methods and constructors of each source file. Within the tokenization, we create for each found method and constructor a separate token file containing the occurred tokens. Furthermore, for each of these files a file with bookkeeping information, in particular the CVE id to later query concrete CVE details, will be created. They also consist of links to code fragments which represent an example patch and their exploit to give developers a better understanding of the weaknesses for patching them afterward. The resulting tokens will be indexed and are the outcome for the preprocessing as well as the bookkeeping CVE information. It has to be applied once each time if the content of the code repository changes.

The Code Processing takes place every time if the source code has been changed. Thereby, the same Parse & Tokenizing and indexing like in the pre-processing will be applied but without adding CVE data to the bookkeeping information. The output of this step are the indexed tokens of source code files.

During the clone detection phase for each code fragment of the security repository and for each method as well as constructor inside of source files will be analyzed whether there are code clones. In detail, the tokens of methods and constructors will be compared. If a match is found, then the checked source code is a code clone of an insecure code fragment, which implies that it potentially also contains a security flaw. These code clones will be interleaved with the CVE data out of the bookkeeping information, which are the results of the vulnerability detection. Thus, should help developers to receive

more knowledge to patch insecure source code fragments. The described approach is visualized in Fig. 3. Later in this chapter, the clone detection will be described in detail.

The effectiveness of this approach relies among others on the data of the reference repository. Therefore, it is inevitable to ensure the adaption and enrichment of knowledge by developers or a repository maintainer. They are able to add new vulnerable, patch and exploit code and modify already stored data.

A big problem for the code clone detection is the time complexity to compute the pairwise similarity for each code fragment combination. Execution time majorly scales with the size of the input precisely of the number of lines of code (LOC) that are processed and searched. For code clone detectors it is prohibited as a time complexity of scalability - $O(n^2)$. If the granularity of the code clone detector is method based, the similarity comparisons increase quadratically with the number of methods. For the SourcererCC various heuristics for reducing the number of similarity computations are described by Sajjani et al. [19]. In comparison of their approach to other state-of-the-art code clone detectors like CCFinderX [8], Deckard [29], iClones [30] and NiCad [31] they reach almost the same time complexity of inputs less than one million LOC. For all bigger input sizes, the SourcererCC has the best execution time. Furthermore, the SourcererCC is the only clone detector of the competing tools that scale to large input sizes of 100 million LOC and is able to consider type 1 to 3 clones.

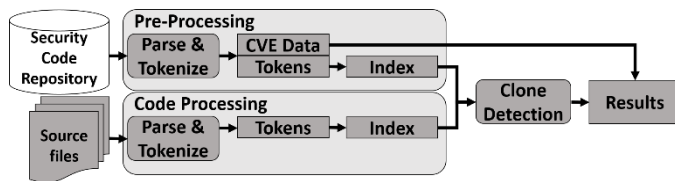


Figure 3. Vulnerability Detection Process

SourcererCC. As a basis for clone detection, we have adopted a state-of-the-art code clone detector named SourcererCC [19]. The detector supports Type 1 to 3 clones and scales to large-scale project repositories while providing high precision and recall. To quantitatively infer if two code snippets are clones a similarity function is applied which returns the non-negative degree of similarity between two code snippets. The higher the value of similarity, the bigger is the likeness between them. This function includes a threshold value ϑ that identifies the lower-bound of the similarity value from which two code fragments count as code clones. In other words, it is a percentage value that represents how many tokens at least should be shared by two code fragments to be identified as code clones. This similarity value is the output of the clone detection process. In the following, the similarity measurement is described formally:

Given two projects P_x and P_y , f as similarity-function and ϑ as threshold, the aim is to find all code block pairs $P_{x,B}$ and $P_{y,B}$ such that $f(|P_{x,B}|, |P_{y,B}|) \geq [\vartheta * \max(|P_{x,B}|, |P_{y,B}|)]$.

Adaptation of SourcererCC. We performed two main adaptations of SourcererCC for our approach:

First of all, we adapted the format of the token files to our needs and implemented a suitable tokenizer; thus its resulting

token files will be interleaved with related information out of the CVE Database. Secondly, we adopted the code clone detector itself to consider only inter-project clones, as discussed later in this section. The main reason for this is that we plan to find clones between an external code repository and a project and not within a single project, like the clone detection is often used for.

Tokenizer. SourcererCC comes with a tokenizer for Java, C and C#. However, for our approach, we needed a method to interleave tokens with CVE meta-information such that clone detection results could provide further security information. Therefore, a custom token format was designed that combines the token information with the additional bookkeeping details, including the CVE id and the metadata of vulnerabilities. This information allows us to trace back from code fragments to the underlying weakness.

To apply the tokenizer a parsing of the source files is needed, such that the tokenizer gets software artifacts of source files like method names and constructors. For exemplary apply and evaluate our approach, we focus on java source code such a java parser is used for parsing code. For each method or constructor, a new list of tokens is created. Whitespaces, operators and comments are ignored. During this procedure, the tokenizer loads the needed metadata out of the security repository. The output of our tokenizer are two files, one including the tokens of a code fragment and the other with the described bookkeeping information enriched with security knowledge.

For further illustration, we present an example method before the preprocessing was applied in Fig. 4.

```

1 public class TokenizerExample
2 {
3     private boolean isEven(int n)
4     {
5         int two = 2;
6         return n % two == 0;
7     }
8 }
  
```

Figure 4. Example: Java Input Source Code for Tokenizer

Figure 5 presents the output of the tokenization of Fig.4.

```

1 {0,private,2,boolean,isEven,int,two,n,return}
  
```

Figure 5. Created tokens of Fig. 4

To complete the tokenization, we count the number of appearances of each token and add them to the output. Hence, a token-file consist of every occurred method name, variable name and its datatypes as well as return values that are named and counted. Regarded to the design constraint for the Java and C# languages, all executable code must be contained inside of method bodies. Therefore, the granularity level for the implemented tokenizer is set to method-based tokenization. This means that only method respective constructors will be processed inside of a class. Imports and class names will be ignored. For other languages like C and C++ it is necessary to

adapt the tokenizer to a class-based tokenization through the absence of this design constraint.

Inter-project clones. Clones can be either intra- or inter-project clones, meaning that the instances of a clone may come either from the same project or from different ones. In our approach, we are only interested in inter-project clones, since we aim to find matches between a given project source code and our security code repository. More formally, let A be the set of source files from the input project and B the set of snippets from the security code repository. Furthermore, let (a_0, \dots, a_n) and (b_0, \dots, b_m) with $n, m \in \mathbb{N}$ code fragments, for which $a_i \in A$ and $b_j \in B$ with $i, j \in \mathbb{N}$. We are interested in pairs of code fragments (a_i, b_j) being code clones.

SourcererCC finds both intra- and inter-project clones, and by default it does not provide a configuration option to deactivate the detection of intra-project clones. Therefore, the *Pre-Processing* phase was added and the inputs for the clone detection was adapted to ignore intra-project clones. Figure 3 presents the modified behavior of the clone detector.

IV. SECURITY CODE REPOSITORY

In our approach, a prerequisite for vulnerability detection is a reference code repository called *security code repository*. This repository contains source code snippets that instantiate already reported weaknesses. Each snippet consists of example code for a vulnerability, its exploit and a patch. We now describe the procedure for creating a security code repository, which we used to test and evaluate our technique. Our process relies on a large-scale repository in which such snippets can be found. To this end, we used GitHub, a suitable repository that is freely accessible to the public. Our process, shown in Fig. 6, contains the four steps *Extract*, *Search& Filter*, *Export*, and *Proof*.

Extract. To find security-related code snippets on Github, adequate terms for the search are needed. We extract them from the CVE database. A possible way to identify a concrete vulnerability is its unique CVE identifier [18]. Therefore, we extract these CVE-ids for the search to find security issues.

Search and Filter. For searching vulnerabilities on Github, the pre-extracted CVE-ids will be used. If a file or a commit within a project matches these identifiers, they will be considered for further processing. To adjust the results of the search to specific programming languages, we defined a file type filter. For example, to restrict the search to Java files, the file endings will be checked for the tag *.java*. Not every content of the found files is security-related. Therefore, a manual prove is necessary to ensure that resulted files contain only security-related code. This procedure is described later in this section.

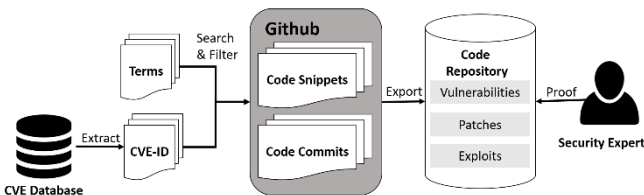


Figure 6. Repository Creation for Reference Code Fragments

Classification. Furthermore, we defined terms to distinguish between the three classes of security-related files; vulnerabilities, exploits and patches. For this classification, we use a simple check whether terms are substrings of texts inside of commits or project descriptions. Examples terms for the class *patches* are fix, solve, update, patch for *vulnerabilities* the term vulnerability and for *exploits*, the substrings are proof of concept (POC) and exploit. We retrieved these terms by the manual unsystematic analyze of founds of the CVE id search received from GitHub. The founds will be automatically classified by these terms into the three mentioned classes.

Export. The classified founds are stored into a code repository with a SQLite database inside of its root. For all matches its related and stored meta-information inside of the CVE will be extracted into the SQLite database. This meta-information are for example the concrete CVE description, the CVE-id and a scoring which represents their characteristics, impact and severity. The vulnerability scoring is based on the Common Vulnerability Scoring System (CVSS) [28].

Proof. As post-processing the repository content has to be manually reviewed to exclude file parts that do not contribute to a vulnerability. If a file with a vulnerability in it is found, often only a part of the file represents a vulnerable code snippet. A security expert has to delete the unaffected methods in these files such that only the critical code remains.

For the creation of a code repository, 20 different weaknesses out of the CVE database were selected. Our previous explained semi-automated tool-based approach found source code examples of exploits, vulnerabilities and its patches assigned to CVE-ids. Thus, security-related code fragments are extracted out of Github. We have reviewed a subset of 102 reported security flaws to check their suitability for representing a vulnerability code snippet that is usable for the approach described within this work. For example, not eligible vulnerabilities can be patched by only importing a newer library version or loading them dynamically by a string literal. Furthermore, code fragments for which weaknesses are spread over multiple methods were partially also ignored. The reason for this is that in the most cases code changes were too small to match the three different types of code clones meaningfully. Table 1 shows the selected vulnerabilities with their associated CVE, their scoring and the affected product, as it is stored into the CVE Database. The created security code clone repository including the SQLite database is uploaded to Github [24].

V. EVALUATION

We evaluated the suitability of the described approach to address our initial research question: *How can we harness available knowledge to facilitate the detection of security vulnerabilities in software code?* To this end, we consider the following evaluation research questions:

RQ1: How many of the vulnerabilities inside of the CVE are attributed to source code?

Not for every vulnerability the reason is a weakness in source code. This question should identify how valuable the focus on source code vulnerabilities is and how many of them could be

TABLE I. CODE REPOSITORY WITH SECURITY-RELATED CONTENT

CVE	Scoring	Product
CVE-2006-2806	7.8	Apache Java Mail Enterprise Server
CVE-2007-5461	3.5	Apache Tomcat
CVE-2007-6203	4.3	Apache HTTP Server 2.0.x and 2.2.x
CVE-2008-2086	9.3	Sun JDK
CVE-2008-5515	5.0	Apache Tomcat
CVE-2009-2693	5.8	Apache Tomcat
CVE-2009-2901	4.3	Apache Tomcat
CVE-2010-4172	4.3	Apache Tomcat
CVE-2011-1475	5.0	Apache Tomcat
CVE-2011-3190	7.5	Apache Tomcat
CVE-2011-3377	4.3	Ubuntu Linux, Opensuse, Redhat Icedtea-Web
CVE-2012-1621	4.3	Apache Open For Business Project
CVE-2012-2459	5.0	Bitcoin und Bitcoin-Qt
CVE-2016-3897	4.3	Google Android
CVE-2016-6723	5.4	Google Android
CVE-2017-0389	7.8	Google Android
CVE-2017-0846	5.0	Google Android
CVE-2017-1217	4.3	IBM WebSphere Portal
CVE-2017-1591	4.3	IBM DataPower Gateway
CVE-2017-9096	6.8	iTextpdf iText

found through using the knowledge of source code of entries stored inside of the CVE by applying the described approach.

RQ2: How accurate is our approach at detecting previously reported vulnerabilities?

We want to check whether it is possible to identify source code reasoned weaknesses through a subset of the reported vulnerabilities stored inside of publicly accessible databases like the CVE.

RQ3: How well does our approach distinguish between vulnerable and patched code fragments?

Sometimes only the change of a few lines of code is necessary to remove a security flaw inside of a code snippet. We investigate on which granularity we can distinguish between patched and insecure code fragments through clone detection.

A. RQ1: How many of the vulnerabilities inside of the CVE are attributed to source code?

First, we investigate how feasible it is to use the information of known and documented vulnerabilities reported in databases like the CVE. For this proof, we check the ratio of entries that could be retrieved through errors within source code to them which do have other origins. The more weaknesses based on source code, the better is the concentration on detecting security flaws within code artefacts. Through this survey the capability of using the content of the publicly accessible database CVE to recognize security issues invoked through source code will be validated. To apply this investigation, the non-code content of the CVE was manually screened. We recognized that in August 2018 only for 62 % of the 103745 CVE entries, a Common Weakness Enumeration (CWE) identifier is assigned.

The CWE compresses different types of weaknesses, which are all identified through a unique CWE-id and categorize different manifestations of vulnerabilities. An example of these types is *CWE-306: Missing Authentication for Critical Function*. All to this type associated CVE entries are

vulnerabilities because of the absence of authentication for critical functions. Therefore, CWE-ids are a well-suited attribute found with our approach. A problem is that there is no identifier for a type that implies all security issues appear within source code. For every set CWE type, it was systematically proved whether it is possible to retrieve out of its description the origin they belong to; induced by source code or others like configurations. A further problem is the absence of assigned CWE-ids for 38 % of the entries of the CVE. The besides without any type information were investigated with the use of their description. Manual checks show that it is possible to find CVEs based on source code via checking their descriptions for the occurrence of substrings. The used substrings were divided into five groups: File Name Endings, Attack Strategies, Configurations, Rejects and Unclassified.

The group File Name Endings contains substrings that represents the data types of programming language source files. The hypothesis is that if a concrete file is addressed within the description of a CVE than their occurrence is provable within source code. Examples of file endings are *.java*, *.cpp*, *etc.*

As attack strategy count for example *Cross-Site Scripting (XSS)*, *Buffer Overflow* *etc.* The idea is to check for the names of attack strategies that maliciously uses a vulnerability occurred in source code like the buffer overflow example, which could be prevented to buffer length checks.

Configuration related vulnerabilities are identified through the occurrence of terms like *config*, *cfg* *etc.* We assume that for CVE descriptions that mention at least one of these terms, their belonging weakness is not detectable via source code analysis.

Rejects are the vulnerabilities that are still remain in the CVE database but were rejected after review. They are marked with the term *Rejected*. This are entries, which could be duplicates or not representing a vulnerability at all.

The left CVE entries that not belong to one of the mentioned groups were assigned to the group Unclassified CVE entries. They will be not considered to answer this research question. The partition of the CVE is summarized in a pie chart in Fig. 5.

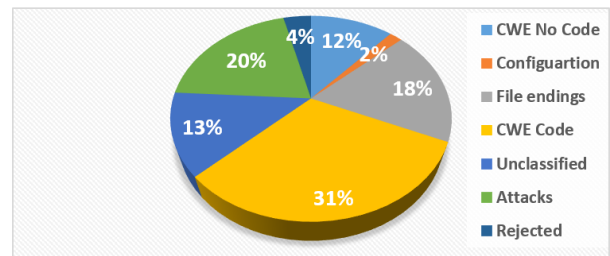


Figure 7. Classification of CVE Database Content

To conclude the results, it is shown that 69 % of the public available vulnerabilities are induced to source code issues. This is composed through the 31% of entries with CWEs that describes security flaws detectable within source code artefacts, the 28 % with a description containing file endings of source code files and the 20 % of CVEs that contain in their description terms of attack strategies uses security flaws within source code. To respond RQ1.1, it is a well-suited strategy to focus on source code for identifying known and reported weaknesses.

B. RQ2: How accurate is our approach at detecting previously reported vulnerabilities?

Experimental Setup

As vulnerability selection for our evaluation, we use the code repository described into Sec. IV. To ensure that not only Type-1 clones are detected but also Type-2 and Type-3 clones we modified each vulnerable code fragment to match the corresponding clone types. For example, all variable names were renamed for Type-2 clones. To obtain Type-3 clones, we removed or added some void statements to Type-1 and Type-2 clones. Our evaluation based on metrics of the information retrieval like *Recall*, *Precision* and *F₁-measure* as described by Manning et al. [15]. We measured the recall based on the chosen vulnerabilities considering the three different types of clones. In this case, we know exactly the number of weaknesses so that the recall can be measured precisely. The precision was measured by a manual validation of the found and highlighted security code clones. To combine precision and recall, we used *F₁-measure*.

The underlying clone detector uses a similarity function inside its clone detection process. This function can be configured by the threshold value θ . Three different θ values were used for the evaluation in combination with the three different types of clones. That means that for every clone type exists three iterations with different thresholds. Beforehand we examined distinct thresholds by hand. On the one hand, lowering the threshold could harm precision. On the other hand, a too high threshold could harm recall. Therefore 3.0, 5.5 and 8.0 were selected as values for θ .

Results and Discussion

Each iteration passes every vulnerability with the given configuration values. The Security Code Clone Detector has flagged all Type-1 clones for the three different θ values. That means that precision, recall, and *F₁* are 100 % for Type-1 clones. The detection rate of Type-2 clones revealed that only eight security flaws out of 20 were detected with a θ of 8.0. The reason is that through the high θ value two code fragments must be too similar to be recognized as code clones. Due to this the higher the value of θ is the bigger the number of similar tokens inside of two code snippets have to be. Thus lead to the low recall for Types-3 clones cause the clones are too different to be code clones of each other.

In contrast, the precision always is 100 %. This could be attributed to the fact that we only focus on finding the three different types of security code clones. The set of data only consist of the given vulnerabilities and not of any secure code fragments. Table 2 summarizes the clone detection results for the security flaws and patches.

TABLE II. EVALUATION RESULTS

θ	Type-1 Clones				Type-2 Clones				Type-3 Clones				Fixes			
	TP	R	P	<i>F₁</i>	TP	R	P	<i>F₁</i>	TP	R	P	<i>F₁</i>	TP	R	P	<i>F₁</i>
3.0	20	100	100	100	20	100	100	100	20	100	100	100	1	100	5	9.52
5.5	20	100	100	100	20	100	100	100	12	60	100	75	3	100	15	26.08
8.0	20	100	100	100	8	40	100	57.14	0	0	100	0	11	100	55	70.97

In the result table TP stands for true positives, R means recall, P means precision and *F₁* stands for *F₁-measure*. The high recall values for all code clone types and θ modifications show that we are able to recognize known and reported weaknesses via our approach. The results for the recall are interleaved with the size of the θ threshold. For small θ values every vulnerability inside of the reference repository is detected, but this leads to a lower precision. Therefore, the answer for RQ2 is that the vulnerability detection for the described approach performs very well with a leak of precision for Type-3 clones.

C. RQ3: How well does our approach distinguish between vulnerable and patched code fragments?

Experimental Setup

In the next step, we have focused on checking precision. Thereto the patched code fragments for the given vulnerabilities were used to examine which security flaws will be detected as vulnerable code clones falsely. In this case, we did not modify the patched code fragments to match each of the three different types of clones but used the code fixes directly as input for the detection process. Patched code fragments distinguish to their vulnerable complement with some changes that removes the weak parts of that fragment. These changes differ in complexity. Some have an extent of multiple lines but other distinguish only by a single line to their vulnerable counterparts. The θ configuration setup for every iteration was the same we mentioned in RQ2.

Result and Discussion

As described in Sec. III, Type-3 clones are hard to detect. The low precision values summarized in Tab. II can be ascribed to the few code modifications inside the patches. Fundamentally, the patches represent Type-3 clones. If a patch has enough code modifications regarding its vulnerability, we can identify it as secure code correctly. If the code changes are too small, it may be flagged as weakness falsely. Relating to RQ3, it is possible to distinguish between vulnerable and patched code fragments as far as enough code modifications are present. On the one hand, the capability to detect code clones of Type-3 increases the amount of secure code, which is falsely identified as insecure. This reduces the precision of the described approach. On the other hand, through code clones of Type-3, the possibility exists to find more code clones of vulnerable code fragments, which increases the recall. Hence our goal is a high recall during detection. Therefore, we accept the false positives within the patch detection.

D. Threats to Validity

For the validity check of our evaluation, we consider the types of threats to validity for empirical software engineering research defined by Wohlin et. al. [26].

Conclusion: Maybe the selection criteria for building the test set influence the results in terms of recall, precision, and *F₁-measure*. To be more precise, it is possible that the size of selected code fragments influence the capability to distinguish between patched and weak code fragments. We have not considered the size of needed patches to close security flaws.

Internal: The used Java Runtime Environment version and the library version of used source code are not considered. Some code fragments are only insecure with specific Java versions or library versions and are secure within other versions. This could result in false positives for the classification of source code fragments as vulnerable that are secure with the used versions. Furthermore, it is imaginable that there are other code clone detection approaches, which tackles the problem of vulnerability detection inside of code fragments better than the chosen one.

Construct: The configuration and the θ adjustment of the code clone detection approach affect the effectiveness of the described procedure. Furthermore, the workflow and the granularity level of the tokenizer influences the results of the code clone detection approach.

VI. RELATED WORK

Vulnerability detection using clone detection. The clone detector ReDeBug [7] is language agnostic and uses a syntax-based pattern matching approach. It can detect some Type-3 clones but cannot detect Type-2 clones respectively clones with slight code modifications. Furthermore one of the design goals was a low false positive rate which harms recall. VulPecker [12] is a system for automatically detecting if a piece of software contains a vulnerability. It consists of a learning phase and a selection algorithm to identify vulnerabilities. This approach can detect Type-1, Type-2 and some Type-3 clones [13] in C/C++ code. CLORIFI [11] combines static and dynamic analysis to detect code clone vulnerabilities. It identifies the security code clones of known vulnerabilities with an n-token algorithm. With the help of concolic testing, CLORIFI tries to reduce false positives by verifying the security flaws. Our work is mostly complementary to the presented ones, as we focus on establishing a right balance between precision and recall, and use a state-of-the-art back-end clone detector that allows us to address scalability and language-independence simultaneously.

Static analysis for security. Our approach can be considered as a static analysis technique that can uncover vulnerabilities without executing the application at hand. Such techniques have been used successfully to uncover vulnerabilities, in some cases better than dynamic techniques such as penetration testing [20]. Most previous techniques are geared to detect specific vulnerabilities based on hard-coded solutions, such as SQL injections [14] and buffer overflows [27]. Fischer et al. [5] have used static analysis to study how severely Android apps are affected by vulnerabilities resulting from copying code snippets from StackOverflow examples. In our previous work [25] we present a tool-based approach that scans imported Java dependencies for known vulnerabilities. It checks the CVE if an entry for the corresponding library exists. Furthermore, we present an approach [3] for maintaining a knowledge base of security knowledge that is used to keep co-evolve the system design after changes in the availability knowledge (e.g., an encryption algorithm previously deemed as secure is broken). However, this work was focused on the design model level rather than on code-level vulnerabilities.

Code clone detection. Different code clone detection approaches with distinct capabilities exist. In their survey, Sheneamer et al. [21] distinguish the following main classes of

approaches. Text-based techniques (e.g. [1]) compares the similarity of code fragments based on terms of textual content. While focusing mostly on Type-1 clones without preprocessing, they are language-independent and easy to implement. Lexical techniques (e.g. [8,19]) divide the input code into a sequence of tokens that are converted into token sequence lines, which are then matched to another. Such techniques can detect various code clone types with higher recall and precision than text passing techniques. Syntactic techniques are either metric-based or tree-based. Tree-based (e.g., [2]) and graph-based techniques (e.g., [22]) parse the code into abstract syntax trees and find cloned code parts using tree-matching algorithms. Metric-based ones (e.g. [17]) compare source code snippet based on metric vectors created for each code snippet. This technique is able to detect Type-1 and Type-2 clones with high time complexity. It reduces the complexity of text-based approaches. Semantic techniques (e.g., [10]) detect two fragments of code that perform the same computation but have differently structured code, which was already named as Type-4 clone.

VII. CONCLUSION

The use of a single vulnerable code snippet can make a whole system insecure [23]. We introduce an approach to prove code fragments of reported vulnerabilities automatically. We detect insecure parts of source code via code clone detection by using code fragments that contain known vulnerabilities. As a result, a developer can be supported in the writing of secure code right from the beginning. With the aid of a database, it is possible to show information about reported security flaws. This information can be used to understand a weakness and, if necessary, provide a patch. The results of the evaluation show that detecting insecure code fragments work very well. We show that there are difficulties in distinguishing between patched and vulnerable source code fragments that are too similar to each other. The capability to detect small differences between them depends on the configuration and the distinction of line circumference from the vulnerable and patched fragments. Our work provides an *Eclipse* plug-in for supporting Java software developers [24]. Furthermore, we investigated the feasibility to distinguish between patched and vulnerable code snippets for our approach. We analyzed the CVE database content to underpin leveraging knowledge of reported vulnerabilities for the detection of security flaws within the source code.

Our future research is striving to compare artificial intelligence approaches in the form of neural networks with the static code clone detection approach described in this work. To solve the problem of our internal validity, we want to enhance the described approach such that the JRE Version of software projects and the recognition of library versions will be considered. Furthermore, a user study about the perception and effectiveness of helping developers and security experts to classify source code fragments is planned. The capability of the code clone detectors in the described approach relies on the data included in the reference code repository. Therefore, we expect to investigate further techniques to enhance the security-related source code within the code repository via leveraging community knowledge. Furthermore, we plan to improve the semi-automatically classification into exploit, vulnerable and patch code fragments.

VIII. ACKNOWLEDGMENT

This work was supported by the German Research Foundation (DFG) under SecVolution (2016 – 2019).

REFERENCES

- [1] Baker, B.S.: On finding duplication and near-duplication in large software systems. In: *Reverse Engineering*, 1995., *Proceedings of 2nd Working Conference on*. pp. 86{95. IEEE (1995)
- [2] Baxter, I.D., Yahin, A., Moura, L., Sant'Anna, M., Bier, L.: Clone detection using abstract syntax trees. In: *Software Maintenance*, 1998. *Proceedings.*, *International Conference on*. pp. 368{377. IEEE (1998)
- [3] Bürger, J., Strüber, D., Gärtner, S., Ruhroth, T., Jürjens, J., Schneider, K.: A framework for semi-automated co-evolution of security knowledge and system models. In: *Journal of Systems and Software* 139, pp. 142{160 (2019)
- [4] Devanbu, P.T., Stubblebine, S.: Software engineering for security. In: Finkelstein, A. (ed.) *Proceedings of the Conference on The Future of Software Engineering*. pp. 227{239. ACM, New York, NY (2000)
- [5] Fischer, F., Böttinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M., Fahl, S.: Stack overflow considered harmful? the impact of copy&paste on android application security. In: *Security and Privacy (SP)*, 2017 *IEEE Symposium on*. pp. 121{136. IEEE (2017)
- [6] Inc., E.: Github (2008), <https://github.com/>
- [7] Jang, J., Agrawal, A., Brumley, D.: Redebug: finding unpatched code clones in entire os distributions. In: *Security and Privacy (SP)*, 2012 *IEEE Symposium on*. pp. 48{62. IEEE (2012)
- [8] Kamiya, T., Kusumoto, S., Inoue, K.: Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering* 28(7), 654{670 (2002)
- [9] Koschke, R.: Survey of research on software clones. In: *Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum für Informatik* (2007)
- [10] Krinke, J.: Identifying similar code with program dependence graphs. In: *Reverse Engineering*, 2001. *Proceedings. Eighth Working Conference on*. pp. 301{309. IEEE (2001)
- [11] Li, H., Kwon, H., Kwon, J., Lee, H.: Clorifi: software vulnerability discovery using code clone verification. *Concurrency and Computation: Practice and Experience* 28(6), 1900{1917 (2016)
- [12] Li, Z., Zou, D., Xu, S., Jin, H., Qi, H., Hu, J.: Vulpecker: an automated vulnerability detection system based on code similarity analysis. In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*. pp. 201{213. ACM (2016)
- [13] Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., Zhong, Y.: Vuldeepecker: A deep learning-based system for vulnerability detection (2018)
- [14] Livshits, V.B., Lam, M.S.: Finding security vulnerabilities in java applications with static analysis. In: *USENIX Security Symposium*. vol. 14, pp. 18{18 (2005)
- [15] Manning, C.D., Raghavan, P., Schtze, H.: *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, United Kingdom (2009)
- [16] Mayer, C.P.: Security and privacy challenges in the internet of things: 158 kb / electronic communications of the easst, volume 17: Kommunikation in verteilten Systemen 2009 (2009)
- [17] Mayrand, J., Leblanc, C., Merlo, E.: Experiment on the automatic detection of function clones in a software system using metrics. In: *icsm*. vol. 96, p. 244 (1996)
- [18] Mitre: Common vulnerability and exposures (1999), <https://cve.mitre.org/>
- [19] Sajjani, H., Saini, V., Svajlenko, J., Roy, C.K., Lopes, C.V.: SourcererCC: Scaling Code Clone Detection to Big-Code. In: 2016 *IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. pp. 1157{1168 (May 2016)
- [20] Scandariato, R., Walden, J., Joosen, W.: Static analysis versus penetration testing: A controlled experiment. In: *Software Reliability Engineering (ISSRE)*, 2013 *IEEE 24th International Symposium on*. pp. 451{460. IEEE (2013)
- [21] Sheneamer, A., Kalita, J.: A Survey of Software Clone Detection Techniques. *International Journal of Computer Applications* 137(10), 1{21 (2016)
- [22] Strüber, D., Acretoiaie, V., Plöger, J.: Model clone detection for rule-based model transformation languages. *Software & Systems Modeling* 18(2), pp. 995{1016 (2019)
- [23] US-Cert: United states computer emergency readiness team (2003), <https://goo.gl/ZCuCc8>
- [24] Viertel, F.P., Brunotte, W., Strüber, D., Schneider, K.: Security Code Repository and Code Clone Detection Eclipse Plug-In (2018), <https://github.com/dev-se/sccd>
- [25] Viertel, F.P., Kortum, F., Wagner, L., Schneider, K.: Are third-party libraries secure? a software library checker for java. In: *The 13th International Conference on Risks and Security of Internet and Systems, CRISIS* (2018)
- [26] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: *Experimentation in software engineering*. Springer Science & Business Media (2012)
- [27] Zitser, M., Lippmann, R., Leek, T.: Testing static analysis tools using exploitable buffer overflows from open source code. In: *ACM SIGSOFT Software Engineering Notes*. vol. 29, pp. 97{106. ACM (2004)
- [28] NIST: Common Vulnerability Scoring System (2005), <https://nvd.nist.gov/vuln-metrics/cvss>
- [29] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *Software Engineering*, 2007. *ICSE 2007. 29th International Conference on*, pages 96{105, May 2007.
- [30] N. Gode and R. Koschke. Incremental clone detection. In *Software Maintenance and Reengineering*, 2009. *CSMR '09. 13th European Conference on*, pages 219{228, March 2009.
- [31] J. R. Cordy and C. K. Roy. The nicad clone detector. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension, ICPC '11*, pages 219{220, Washington, DC, USA, 2011. IEEE Computer Society.

Case-Based Cybersecurity Incident Resolution

Marcelo Colome, Raul Ceretta Nunes, and Luis Alvaro de Lima Silva

Graduate Program in Computer Science - PPGCC

Applied Computing Department, Federal University of Santa Maria

Av. Roraima, Campus UFSM, Camobi, Santa Maria, RS, Brazil

{marcelocolome, ceretta, luisalvaro}@inf.ufsm.br

Abstract - Intelligent computing techniques have a paramount importance to the treatment of cybersecurity incidents. In such Artificial Intelligence (AI) context, while most of the algorithms explored in the cybersecurity domain aim to present solutions to intrusion detection problems, these algorithms seldom approach the correction procedures that are explored in the resolution of cybersecurity incident problems that already took place. In practice, knowledge regarding cybersecurity resolution data and procedures is being under-used in the development of intelligent cybersecurity systems, sometimes even lost and not used at all. In this context, this work proposes to integrate Case-Based Reasoning techniques and IODEF standard in order to retain concrete problem-solving experiences of cybersecurity incident resolution to be reused in the resolution of new incidents. Experimental results so far obtained with a Case-based Cybersecurity Incident Resolution System (CbCSecIRS) implemented show that information security knowledge can be retained in a reusable memory, so improving the resolution of new cybersecurity problems.

Keywords— *Cybersecurity incidents; case-base reasoning; information security.*

I. INTRODUCTION

Information security issues have a critical impact on business mainly because the treatment of security incidents is highly expensive and time consuming to organizations. According to the ISO/IEC 27035 [1], processes of information security management should be grounded on approaches to the capture, structuring, and dissemination of security knowledge in each part of the organization. Fundamentally, security knowledge is expressed as various kinds of lessons learned constructed and refined over the time by cybersecurity experts about how to identify and treat cybersecurity incidents. As investigated here, this knowledge must be retained and reused systematically so that cybersecurity problems could be effectively approached. When such knowledge-based solutions are reused, for instance, the cost of reapplying them (instead of reconstructing them from scratch) each time a cybersecurity incident occurs can be reduced significantly. In crisis situations due to the occurrence of cybersecurity incidents, the collection and representation of incident resolution (CSecIR) procedures is fundamental to organizations since they can be revisited by security analysts as to promptly and comprehensively approach the treatment of cybersecurity issues.

The use of central systems in the collection, correlation, and analysis of data related to security incidents is a common practice in many countries, where Computer Emergency Response Teams (CERT) commonly detect and report thousands

of cybersecurity incidents a year. Each time an incident is reported by the CERT, it should be analyzed and solved by the Computer Security Incident Response Team (CSIRT) which is in charge of managing the computer network where the incident took place. To help the sharing of security information exchanged between CSIRTs or other operational security teams, the Internet Engineering Task Force (IETF) has proposed the Incident Object Description Exchange Format (IODEF) [2], which is a format directed to the broad representation of computer security information. Moreover, an IODEF extension aiming to facilitate the representation and exchange of enriched cybersecurity information was also proposed [3, 4]. Despite these efforts, it is still challenging to reuse security solutions [5, 6], especially those derived from concrete experiences of cybersecurity incident problem-solving.

In Artificial Intelligence (AI), while machine learning algorithms explored in the cybersecurity domain are aimed at presenting reliable intrusion detection solutions, these algorithms seldom approach the representation and reasoning with cybersecurity incident resolution procedural knowledge. In practice, such cybersecurity knowledge is being under-used in the development of intelligent cybersecurity systems, decreasing the effectiveness of Cybersecurity Incident Resolution Systems (CSecIRS). With the help of the Case-Based Reasoning (CBR) techniques [7], this paper approaches the collection and representation of this knowledge in the form of cases. Importantly, such cybersecurity incident resolution cases can be shared and reused as part of fundamental case-based knowledge management tasks [8, 9]. In this context, this paper shows how to build Case-based Cybersecurity Incident Resolution Systems (CbCSecIRS) based on information security attributes detailed according to the IODEF standard, including its cybersecurity extension. Instead of acting as a single cybersecurity solution, the overall idea of following the IODEF pattern is to permit to integrate the CbCSecIRS representation and reasoning capabilities from both intrusion detection systems and cybersecurity incident resolution systems.

The paper is structured as follows: Section II describes how cybersecurity incidents are approached and Section III presents related works where CBR techniques are explored in the cybersecurity domain. While Section IV presents our CbCSecIRS proposal, Section V describes experiments and results so far developed in our project. Finally, conclusions are presented in Section VI.

II. THE RESOLUTION AND REPRESENTATION OF CYBERSECURITY INCIDENTS

Knowledge regarding the resolution of cybersecurity incidents is a crucial asset to organizations. To be competitive, large amount of resources are being invested by security companies in order to not lose their valuable cybersecurity incident resolution experiences. By maintaining such lessons learned in a reusable memory, security analysts have the means of avoiding the costly reconstruction of “new” security solutions each time a cybersecurity incident problem occurs. Although large amount of data about incidents is being collected and explored by security companies via different AI approaches, the ISO/IEC 27035 standard [1] states that the processes of cybersecurity incident treatment can be organized in different activities: *i) Plan and prepare*: aim to develop incident treatment plans, check-lists of tasks to be executed when such cybersecurity threads occur, and communication plans aiming to record information about how entities involved should be prepare to communicate in the occurrence of security calamities; *ii) Detect and report*: as recommended in [10], multiples forms of reporting the cybersecurity incidents should be explored. In addition to the manual reporting, cybersecurity incidents can be reported automatically by security services or other entities as CERTs; *iii) Evaluate and decide*: the concrete occurrence of the cybersecurity incident should be evaluated, as well as the magnitude and consequences of such incident. Once this evaluation is developed, the origin of the cybersecurity incident can be traced properly; *iv) Respond*: involves the incident treatment actions that are properly planned in advance. Based on such treatment plans, recommended problem-solving steps aimed to deal with the cybersecurity incidents are executed. It means that appropriate resolution actions should be taken as to recover from the cybersecurity incident, in addition to incident documentation and communication to stakeholders; *v) Record*: the recording of the lessons learned should start as soon as the cybersecurity incident is closed. In doing so, this recording aims to assess whether the solution designed by the CSIRT was successful. An important task here is to document the cybersecurity incident, including not only its categorization but also its procedures of treatment.

In this paper, the techniques proposed are concerned with the outputs of the *detect and report* activities, retrieving past cybersecurity incident solutions that are relevant to the development of *evaluate and decide* activities. Then, cybersecurity incident resolution plans retrieved are used in *respond* activities, permitting to construct new plans to be explored in the *record* activities. So, a typical problem in such cybersecurity incident resolution scenario is the maintenance of lessons learned. We highlight such lessons are not only captured by the recording of factual information of cybersecurity incidents. In practice, alternative machine learning techniques can be successfully explored in the learning of how to automatically detect cybersecurity threads from such factual data. What we highlight in this work is that these lessons are also formed by the treatment procedures used by security analysts in the resolution of cybersecurity incident problems. So, this concrete experience-based knowledge ought to be collected and stored so that it can be shared among different security systems, in

addition of being queried and reused as to better solve new cybersecurity incidents.

In the processes of cybersecurity incident treatment, the IODEF standard defines a *data format* directed to the representation and exchanging of information about cybersecurity incidents [3, 4]. The IODEF data model includes data about hosts, networks and services; attack methodologies and forensic pieces of evidence; incident impact; and approach to document the cybersecurity investigation and treatment workflow. This standard also provides a framework to share the incident information that is usually exchanged by CSIRTs as to facilitate the machine-processing of such information. In essence, the IODEF data format is organized in set of data classes, derived from a basic class *Document* that contains one or more *Incident* class. Each aggregated Incident class describes in its derived classes commonly exchanged information when reporting or sharing derived analysis from security incidents. The cybersecurity incident IODEF extension [3, 4] increased Incident class representation capabilities. Despite the large number of resources provided by the IODEF, as it was developed to be adaptable to the different organizational needs, the classes that are required to represent a cybersecurity problem are of particular importance as this paper shows how a CbCSecIRS can explore them in the representation of concrete experiences of cybersecurity resolution problems (details in the section IV).

III. CASE-BASED REASONING IN THE CYBERSECURITY DOMAIN

In AI, Case-Based Reasoning [7] relies on a lazy-learning approach to machine learning which focuses the resolution of new problems by reusing solutions recorded in past problem-solving experiences represented as “cases”. Given a new problem to be solved as a query in such CBR systems, the key problem-solving steps are 1) the retrieval of similar cases from a case base, 2) the reuse of solutions recorded in the most similar cases retrieved, 3) the revision of such retrieved solutions as to deal with possible differences between past and new case situations and 4) the retention of new case-based problem-solving experiences in the case base as a way of learning how to solve new problems. Relevant works with CBR in cybersecurity research context follow.

In [11], a CBR system explores the organization of attack cases, where a hierarchical structure containing attributes from possible attack situations is used in the representation of such problem cases. To detail the solutions of such cases, the textual description of countermeasures and the user satisfaction degree for solution proposals are used. Although this work presents a relevant solution for this cybersecurity knowledge management problem, it only approach a limited set of response types to incidents.

With the use of CBR, [12] details a RFM (*Recency, Frequency, Monetary*) technique aimed at reducing false alerts. Considering how recent the security event occurred, its frequency and attributes values, this approach relies on the statistical analysis of log files to detect anomalies. Then CBR is applied on the identification of attack patterns that are similar to past ones. This work is also focused on the incident detection and determination of security event responses, where such responses are expressed as commands to computer security

services. However, this work does not explore the collection and representation of response plans to the treatment of cybersecurity incidents.

In [13], ontologies are integrated to CBR techniques in order to construct a decision-making and response system to the treatment of cybersecurity incidents. In particular, the ontology model is used in the standardized representation of such incidents, resulting on a hierarchical organization of attack types. While this work does not follow cybersecurity representation standards, the collection of automated attack information and manual attack information are the inputs of the resulting CBR system.

In [14], a CBR system to support the construction of cybersecurity incident responses is described. Using information from past attack cases, this system classifies new attacks to better maintain a secure network. While each attack is represented by a sequence of events, each response is represented by a partially ordered set of resolution actions. These attacks are compared with past attack cases stored in a case base, allowing the reuse of response plans recorded as a solution to the new attack situation. Although this work considers the determination of responses to cybersecurity incidents, it is mostly focused on the incident detection through CBR.

From such works, it is possible to state that the exploration of CBR techniques in the cybersecurity domain is limited and the benefits due to the integration of such AI technique with cybersecurity data standards are still open to investigation. Relying on the proposal of a CbCSecIRS proposal, this paper aims to further approach this gap.

IV. A CASE-BASED REASONING MODEL FOR CYBERSECURITY INCIDENT RECORDING AND RESOLUTION

The recording and reasoning with expert knowledge regarding to the resolution of cybersecurity incidents is crucial to the effective treatment of new incident problems. In our Case-based Cybersecurity Incident Resolution System (CbCSecIRS) this knowledge is approached as concrete experiences of problem-solving modeled as *cases*. Once such cases stored in a *case base* are available for similarity-based computations, detailed experience-based answers to the resolution of cybersecurity incidents can be better reused by security analysts. In practice, concrete cybersecurity incidents are recorded in a shared memory, allowing security teams to maintain reusable security treatment knowledge.

To allow cybersecurity incident cases (represented as problem-solution pairs) to be reused, the first modeling task is to represent the problem (incident) according to the IODEF standard. In this way, such incident representation is in conformity with other security proposals directed to the improvement of the operational capabilities of CSIRT teams [3, 4]. Once the incident representation complies with IODEF standard, the CbCSecIRS can communicate with other security systems to allow the acquisition/exchange of cybersecurity incident cases (i.e. problem part of such cases). In addition, security logs received along with incident descriptions can also be examined by security analysts as part of the case acquisition and representation tasks.

The case-based process of cybersecurity incident treatment starts when incidents represented in IODEF are captured by the security analysts. Using the CbCSecIRS, concrete occurrences of new cybersecurity incidents are taken as queries. Once retrieved cases (similar to the current incident situation) are available for examination, the incident treatment plans recorded in the cases retrieved can be re-executed. When such proposed solutions prove to be effective in the resolution of the current problem, such new experience of problem-solving can be recorded in the case base as part of a continuous improvement of the case knowledge which is maintained by the system. If there isn't a good solution and a new resolution is planned and executed, it also can be recorded in the case base. Such recordings allow the CbCSecIRS to dynamically learn new cases as to augment its capabilities of solving cybersecurity incidents.

A. The Case Base Modeling

In the modeling of a case, an incident (problem) is represented by a set of attributes and values along with the incident resolution (treatment plan) expressed by a set of actions. Each incident presents particular behaviors and requires particular attributes to be recognized. Thus, a cybersecurity incident in the CbCSecIRS is modeled by incident type, where types considered in our project are listed in Table I.

TABLE I. INCIDENT TYPES MODELED IN THE CASE BASE

Type	Description
<i>Bot</i>	An organization asset starts to be part of a malware infected computer network. The computers of this network are controlled by hackers (<i>botmasters</i>)
DoS	Deny of service attack. An inundation attack against a target (host or service) to turn it unavailable. This cybersecurity incident can be centralized or distributed (DDoS)
<i>Proxy</i>	A <i>proxy</i> server is infected in order to make anonymous the hackers that are using it. So, such anonymous hackers use the proxy server to make other attacks
<i>MaliciousURL</i>	It is a computer storing malicious files which are accessible by a URL
<i>Copyright</i>	A <i>host</i> shares or received protected material by copyright
<i>Spam</i>	Unsolicited message sent from a host to other users
<i>Scan</i>	A <i>host</i> scans other host ports in order to find vulnerabilities that may allow an attack
<i>LoginAttempt</i>	Login attempts by brute force in a service account. The overall aim is to obtain an un-authorized access on the system
<i>Phishing</i>	It is an attempt of deceive a legal user using a fake web page with is similar to a correct one
<i>Defacement</i>	Content modification of legal web site without authorization

The attribute selection by incident type derived from the incident characterization detailed in [11-14]. After the identification of such set of attributes from literature, its consistency was checked against the cybersecurity incident reported by the

Brazilian academic network CSIRT. While there are attributes that are common to different types of cybersecurity incidents, others are specific to one type. As a result, eight common attributes and twelve specific attributes were detected and selected to model the incidents in a case. Despite our selection, we highlight the expert can include others when necessary.

To represent the incident case, the modeled cybersecurity incidents were mapped to IODEF format. Figure 1 illustrates how the standard IODEF classes were adapted to support our case model. The *Incident* class derives from *IODEF-Document* class. It is mandatory in IODEF format. The *IODEF-Document* class contains the attributes *version* and *lang* that according to RFC4646 [15] must ever be filled. The *Incident* class expresses a standardized description of commonly shared incident attributes. It specifies the time the incident is reported (*DetectTime*) along with a textual description of the incident (*Description*). The *purpose* attribute is mandatory and it is used to express the reason by which the IODEF document was created (traceback, mitigation, reporting, other). The *Flow*, *System*, *Node*, *Address*, and *OperatingSystem* classes describe environment features involved in the cybersecurity incident. The *Method* class describes the method used in the attack and its derived *Reference* class makes reference to vulnerabilities, alerts from IDSs, data about malwares, and other information from the IODEF cybersecurity extension format. The *Service* and *Application* classes describe details about attributes related to resources involved in the incident. Finally, *AdditionalData* class is included to extend the IODEF model, representing different attributes like Logs, HashFromMalware, Agent, Title, Size, IpCC, IpOrigin, TtConnections, ProxyType.

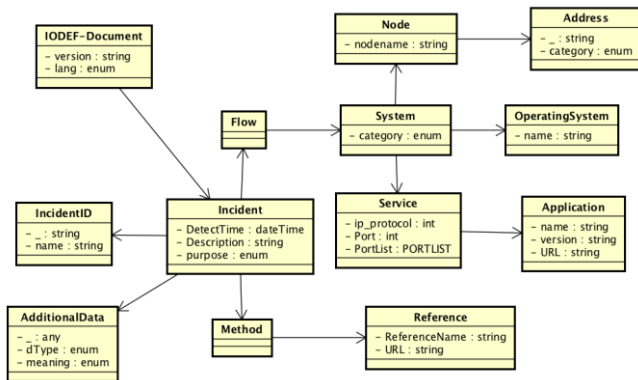


Fig. 1. The IODEF representation used by incident resolution cases.

B. Resolution of Cybersecurity Incidents

The cybersecurity incident experiences of problem-solving retrieved from the case base ought to be the most similar cases to the current problem. As implemented in the CbCSecIRS, this similarity is indicated by a numerical value between 0 and 1, where 1 is the highest similarity between two cases. The similarity computation is developed by comparing n pairs of a_i and b_i attributes represented in the case structure. Once such local similarities (similarities between attributes) are computed, a global similarity (similarities between cases) is measured. To compute this global similarity, an aggregation functions make use of weight values associated to each attribute

used in the similarity computation. As described in the Equation (1), these weight value W_i represent the relative importance of the i attributes in the solution of the problem. Based on this similarity assessment, the resulting similarity computation indicates how similar the cases a and b are.

$$sim(a, b) = \sum_{i=1}^n W_i \times sim_i(a_i, b_i) \quad (1)$$

To compute the distance between two cases, the Euclidean distance function is used. The solution of a cybersecurity incident problem involves the characterization of a problem situation and the consequent selection and execution of a set of actions/procedures directed to the correction (mitigation) of the problem. In this work, these actions are recorded in body of cases as simplified plan-like structures of incident treatment. Figure 2 illustrates a plan constructed by security analysts from the security division of a commercial data center to approach a Bot incident type. In practice, this plan details a cybersecurity resolution script that is followed by these analysts when they need to treat a cybersecurity incident situation.

Incident Response Plan

Incident #997164

1. Quickly disable the host access to the institution's network
2. Open a new ticket to inform the incident to the User Support Center in order to send a technician to the computer's place
3. Analyze evidence to identify the incident
4. Execute a complete scan using the antivirus software
5. If some infected file was found, follow the instructions prompted by the antivirus software
6. If the antivirus software cannot be executed, reboot the computer in "safe mode" and repeat the steps 4 and 5 of this plan and then reboot the computer in "regular mode"
7. If the antivirus software can't be executed in "regular mode" or "safe mode", use the specific tool for the infected file removal
8. If some Operation System file is infected, reinstall the Operation System
9. Ensure that the Operation System is working with the most recent updates, and the updates are set to be automatic
10. Enable the access of this host to the network of the organization
11. Ensure that the computer's firewall is installed with the most recent updates
12. Ensure that the antivirus is installed with the most recent updates
13. Recommend the user to follow orientations from the Internet Security Document available at <https://cartilha.cert.br>
14. Contact the Incident Security Center in order to inform them about the response given to this incident

Fig. 2. Response plan used in the treatment of a Bot incident.

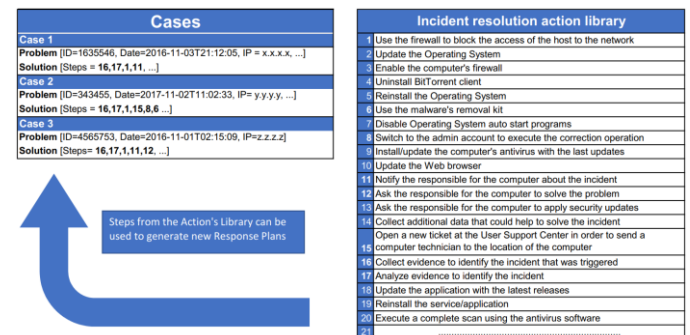


Fig. 3. Incident cases represented according to a cybersecurity incident resolution action library.

To standardize the description of cybersecurity incident resolution plans, a set of actions was represented in a library. So, resolution actions are reused from this repository in the specification of treatment plans for different kinds of cybersecurity incidents. For instance, Figure 3 presents three different cases in which their respective treatment plans were detailed according to plan step indices defined in the library (labeled according to such indices). In practice, the library reflects the steps used in the treatment of the cybersecurity incidents stored in the case base of the CbCSecIRS.

In our project, the CbCSecIRS implemented the K-Nearest Neighbours algorithm, where a (weighted) Euclidian distance function was used in the computation of case similarities, and the consequent retrieval of cases from the case base as to provide cybersecurity treatment answers to incident situations detailed as queries.

V. EXPERIMENTS AND RESULTS

Experiments were developed as part of the evaluation of the CbCSecIRS approach proposed in this work. The goal was twofold: first, to assess the reuse of past experiences of cybersecurity incident problem-solving in the resolution of new problems in this cybersecurity domain and, second, to assess the accuracy of the CbCSecIRS implemented. To approach these goals, a set of 259 cybersecurity incidents used in the experiments were collected from the security division of a commercial data center.

To approach the first experimental goal, new cybersecurity incident situations were collected and used in the tests: the cybersecurity incidents number 2102389 and 2261674 (these are solved cybersecurity incident problems by different participants of the security team of the company, although they were not known during the system development). Each one of these new case problems was expressed as a query in the CbCSecIRS, allowing one to retrieve the most similar cases to them from the case base. In many senses, the aim was to examine if the cybersecurity resolution procedures recorded in the retrieved cases could be reused on the treatment of the current problem. In doing so, the retrieved cases for each executed query were presented to a security expert from the commercial data center organization. Whenever possible, this expert offered positive feedback when the resolution plan retrieved could be properly reused on the treatment of the current problem situation. An example of such research in action case study is presented in Figure 4.

Incident			
ID 2102389			
Incident	Solution steps		
DetectTime	2017-05-09 14:44:30	1	16
IP	21	2	17
Logs	...	3	28
Description	...	4	29
Type	Bot	5	8
RefID	-	6	6
Category	Source	7	9
Port	34934	8	2
Hostname	26	9	3
IpCC	65	10	31
TiConnections	-	11	38
Protocol	http	12	23
MalwareName	Downadup	13	-
RefURL	36	14	-

Recommended solution #1			
ID 1483711			
Incident	Solution steps		
DetectTime	2016-09-01 12:21:31	1	16
IP	23	2	17
Logs	...	3	28
Description	...	4	29
Type	Bot	5	8
RefID	-	6	6
Category	Source	7	9
Port	34590	8	2
Hostname	28	9	3
IpCC	65	10	31
TiConnections	-	11	38
Protocol	http	12	23
MalwareName	Downadup	13	-
RefURL	36	14	-

Recommended solution #2			
ID 1510754			
Incident	Solution steps		
DetectTime	13-09-16 13:20:00	1	16
IP	23	2	17
Logs	...	3	28
Description	...	4	29
Type	Bot	5	8
RefID	-	6	6
Category	Source	7	9
Port	42247	8	2
Hostname	28	9	3
IpCC	65	10	31
TiConnections	-	11	38
Protocol	http	12	23
MalwareName	Downadup	13	-
RefURL	36	14	-

Fig. 4. Incidents number 2102389, 1483711 and 1510754.

In Figure 4, the 2102389 incident was used as a query in the CbCSecIRS, allowing one to retrieve the 1483711 and 1510754 incidents from the case base. All these incidents were characterized as Bot types. These retrieved cases have treatment plans that were considered similar to the plan recorded in the query case. So, the CbCSecIRS was successful on the resolution of this 2102389 test case, showing that the proposed technique was able to maintain the cybersecurity incident resolution knowledge to this kind of problem.

Another example is presented in Figure 5. To the 2261674 incident used as query, the 1022675 and 1620589 cases were retrieved from the CbCSecIRS case base. Both retrieved cases were of the *Copyright* type detailing the illegal sharing of movies in the *BitTorrent* platform. In relation to the treatment plan represented in the retrieved cases, only the 1022675 case contained a highly similar treatment plan in relation to the plan recorded in the query case. Although a solution to the 2261674 query situation could be obtained with the reuse of the plan recorded in the most similar case retrieved, the 1620589 case recorded a new kind of treatment in relation to the other cases considered. Figure 5 presents these cybersecurity incident treatment plans side-by-side, allowing one to observe that the 2261674 incident contained more detailed resolution steps than the more general resolution ones represented in 1620589 case. It means that the retrieved solution could not be fully reused in the solution of the test case situation. That was because it was necessary to develop more particular resolution actions in the treatment of the current problem situation. All in all, as part of traditional knowledge acquisition and representation tasks, improvements in the ways cybersecurity resolution procedures are represented in cases still have to be applied in the CbCSecIRS proposal.

IdIncident	2261674	1620589
Step 1	Collect evidences to identify the incident that was triggered	Collect evidences to identify the incident that was triggered
Step 2	Analyze evidences to identify the incident	Analyze evidences to identify the incident
Step 3	Use the firewall to block the access of the host to the network	Use the firewall to block the access of the host to the network
Step 4	Open a new ticket at the User Support Center in order to send a computer technician to the location of the computer	Notify the administrator of host/network/server/webpage about received incident
Step 5	Uninstall the client of BitTorrent protocol	Request the administrator of the host/network/server/webpage to block the communication with the network
Step 6	Guide the user to follow the published security tips	After incident treatment, request access
Step 7	After incident treatment, request access	After incident resolution, unblock the access in the firewall
Step 8	After incident resolution, unblock the access in the firewall	Notify the CAIS about incident resolution
Step 9	Notify the CAIS about incident resolution	

Fig. 5. Incident resolution plans for cases 2261674 and 1620589.

In addition to such research in action case study experiments, tests aiming to evaluate the CbCSecIRS accuracy were developed as part of the second experimental goal. To do so, the cases in the case base were randomly divided in p partitions of equal size, where $p = 10$. Then, a *K-Fold Cross Validation* technique was used in the evaluation of the system accuracy. In different test runs, for instance, the cases belonging to one of these partitions were used as query cases, while the remaining cases were maintained in the case base so that they could be retrieved as solutions for such a query. In case the retrieved cases and the query cases contained similar cybersecurity incident resolution plans, the answer generated by the system was considered correct. Otherwise, the system offered an incorrect answer to the current problem situation. In

a first run, tests were developed using a similarity function in which a weight = 1.0 was attached to all case attributes being used in the similarity computations, indicating that such attributes have the same importance in such computations. In a second run, the weight values for such case attributes were adjusted according to the opinion of a cybersecurity domain expert from the commercial data center organization.

Table II shows the accuracy results obtained when the K-Fold Cross Validation technique was executed. Although considering different similarity thresholds in the retrieval algorithm used by the CbCSecIRS (95% and 60% minimal similarities), these accuracy results were positive (i.e. as good as to accuracy results presented by other works in this application domain [11-14]) when adjusted weight values were used and when all weight values were equal to 1.0 in the similarity function used by this system.

TABLE II. THE ACCURACY OF THE CBCSECIRS

	1-NN	2-NN	3-NN	4-NN	5-NN
Similarity threshold = 60%, weights w = 1	87.50	84.38	88.89	83.33	80.00
Similarity threshold = 95%, weights values determined by a domain expert	93.33	90.00	95.24	91.67	90.00

VI. CONCLUDING REMARKS

Organizations spent a lot of time and money on the treatment of cybersecurity incidents due to the fact that it is still challenging to maintain their concrete experiences of cybersecurity problem-solving. To approach this problem, this work describes the knowledge acquisition and representation activities that cybersecurity system developers can explore when building CbCSecIRS. In doing so, the cybersecurity incident case model used by these CbCSecIRSs is based on attributes detailed in the IODEF standard. Instead of acting as an isolate cybersecurity solution, the overall idea of following the IODEF standard is to permit to integrate the reasoning capabilities from both intrusion detection systems and cybersecurity incident resolution systems.

As discussed in this work, the CbCSecIRS offers the capability of retrieving cybersecurity incident data and incident resolution procedures represented in cases. Such cybersecurity knowledge is are organized and specified explicitly in the case structure, allowing to be reused by security analysts in different cybersecurity problems. In particular, cybersecurity knowledge regarding incident resolution actions now recorded in cases amount to a concrete explanation about how to better approach those kinds of problems. This explanation capability is crucial when cybersecurity emergency circumstances occur (i.e. after an attack happened, even in face of protection barriers). That is because security analysts are required to promptly and effectively explain their actions in such crisis situations as to mitigate the damage that a cybersecurity event very often causes in the computer infrastructure of an organization.

The CbCSecIRS proposal detailed in this work can have a dual application since it can be explored in both the cybersecurity incident detection and the cybersecurity incident resolution. In practice, cybersecurity incident cases do express incident resolution knowledge which can complement the func-

tionalties required to automatically detect and prevent those incidents as explored by other AI techniques in the cybersecurity domain. Although the experiments presented here can be expanded in different ways, the results show a positive scenario in which our CbCSecIRS proposal is relevant for cybersecurity analysts because it accurately relies on similarity-based computations to connect incident detection data with incident resolution procedures which can now be maintained in the structure of reusable cases.

REFERENCES

- [1] ISO/IEC, "ISO/IEC 27035:2016, Information technology - security techniques - information security incident management," Int. Organization for Standardization, 2016.
- [2] R. Danyliw, "RFC 7970: The Incident Object Description Exchange Format Version 2," Internet Engineering Task Force (IETF), 2016.
- [3] T. Takahashi, K. Landfield, and Y. Kadobayashi, "RFC 7203: An Incident Object Description Exchange Format (IODEF) Extension for Structured Cybersecurity Information," Internet Engineering Task Force (IETF), 2014.
- [4] T. Takahashi, and D. Miyamoto, "Structured cybersecurity information exchange for streamlining incident response operations," in NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, Istanbul, Turkey, 2016, pp. 949-954.
- [5] H. Gascon, B. Grobauer, T. Schreck, L. Rist, D. Arp, and K. Rieck, "Mining attributed graphs for threat intelligence," in Seventh ACM on Conf. on Data and Application Security and Privacy (CODASPY '17), Scottsdale, Arizona, 2017, pp. 15-22.
- [6] M. B. Line, I. A. Tøndel, and M. G. Jaatun, "Current practices and challenges in industrial control organizations regarding information security incident management – Does size matter? Information security incident management in large and small industrial control organizations," *Int. Journal of Critical Infrastructure Protection*, vol. 12, pp. 12-26, 2016.
- [7] R. L. d. Mantaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. L. Maher, M. T. Cox, K. Forbus, M. Keane, A. Aamodt, and I. Watson, "Retrieval, reuse, revision and retention in case-based reasoning," *The Knowledge Engineering Review*, vol. 20, no. 3, pp. 215-240, 2005.
- [8] K. D. Althoff, and R. O. Weber, "Knowledge management in case-based reasoning," *The Knowledge Engineering Review*, vol. 20, no. 3, pp. 305–310, 2005.
- [9] K. Dalkir, and J. Liebowitz, *Knowledge Management in Theory and Practice*: The MIT Press, 2011.
- [10] S. Metzger, W. Hommel, and H. Reiser, "Integrated security incident management – concepts and real-world experiences," in Sixth Int. Conf. on IT Security Incident Management and IT Forensics, Stuttgart, Germany, 2011, pp. 107-121.
- [11] F. Jiang, T. Gu, L. Chang, and Z. Xu, "Case Retrieval for Network Security Emergency Response Based on Description Logic," in 8th Int. Conf. on Intelligent Information Processing (IIP), Hangzhou, China, 2014, pp. 284-293.
- [12] H. K. Kim, K. H. Im, and S. C. Park, "DSS for computer security incident response applying CBR and collaborative response," *Expert Systems with Applications*, vol. 37, no. 1, pp. 852-870, 2010.
- [13] L. Ping, Y. Haifeng, and M. Guoqing, "An incident response decision support system based on CBR and ontology," in Int. Conf. on Computer Application and System Modeling (ICCASM 2010), Shanxi, Taiyuan, 2010, pp. 337-340.
- [14] G. Capuzzi, L. Spalazzi, and F. Pagliarecci, "IRSS: Incident Response Support System," in Int. Symposium on Collaborative Technologies and Systems (CTS 2006), Las Vegas, NV, USA, 2006, pp. 81-88.
- [15] A. Phillips, and M. Davis, "RFC 4646: Tags for Identifying Languages," Network Working Group, 2006.

Verifying Static Aspects of UML models using Prolog

Feng Sheng Huibiao Zhu* Zongyuan Yang Jiaqi Yin Gang Lu*

School of Computer Science and Software Engineering

East China Normal University, Shanghai, China

Abstract—The Unified Modeling Language (UML) provides a number of diagrams to describe the modeling system from different perspectives, which contain overlapping information about the systems. However, it does not provide any means of meticulously checking consistencies among the overlapping elements. In this study, we propose an approach for consistency checking of UML class diagrams and object diagrams using Prolog. First we formalize the model elements based on metamodel and convert the models into Prolog facts. Then we define some consistency rules that are encoded into Prolog. The Prolog's reasoning engine automatically checks the consistencies of models. In addition, we provide interfaces to query models for properties, elements and submodels. The design errors can be effectively avoided and the correctness of code-generalization can be guaranteed according to our approach.

Index Terms—Class Diagram, Object Diagram, Consistency Checking, Prolog

I. INTRODUCTION

The Unified Modeling Language [1] has been developed as a standard object-oriented modeling notation in Model Driven Engineering (MDE) and is widely used in industry. It provides numbers of diagrams to model different aspects of the systems, such as the static views (class diagrams, object diagrams) and dynamic views (sequence diagrams, statecharts). In addition, it offers a variety of tools that cover all the features of the system modeling for a more complete description of the models. However, the syntax and semantics of the UML are semi-formally defined in terms of a metamodel combining natural language descriptions, UML notations and Object Constraint Language (OCL), which is not sufficient to express the semantics of the UML models precisely. Moreover, UML uses the different models to characterize the same system from different perspectives. Change in one diagram may ultimately affect the other diagrams, and result the inconsistencies between different diagrams.

UML models can be represented in the form of a theory in mathematical logic, such as the description logic [2], data refinement [3] and category theory [4]. By transforming UML models into mathematical logic, the problem of inconsistencies can be regarded as the problem of contradictions in the logic theory. A system is consistent if it does not contain any contradictions. Inconsistencies in UML models reveals design errors in software development. Moreover, the inconsistencies will

not be propagated to the codes if we perform the consistency checking at design phase.

Various approaches [6] [13] [15] [16] have been proposed to check the consistency in UML. Typically, approaches devoted to the verification of UML models convert the models into formal semantic domains. Besides, different types of consistency rules are defined on the basis of these conversions. The models' consistencies are verified according to the possibility that the models satisfy these consistency rules. Straeten et al. [2] developed an extension of UML metamodel and presented a classification of inconsistency problems using description logic. They expressed the detection and resolution of consistency conflicts by means of rules. Egea and Rusu [11] demonstrated the structural and semantic conformance between models and metamodels by transforming the models into Maude. Besides, the satisfiability modulo theories (SMT) is often used to support the consistency verification of UML models. Soeken et al. [8] presented an automatic approach that checks the verification for the UML dynamic models. The underlying verification problem is encoded as an instance of the satisfiability problem and subsequently solved using a SAT Modulo Theory solver. However, the SMT solvers typically only support decidable theories and are not sufficient for consistency checking. Storrie [12] proposed a representation of models based on Prolog. He provided query interfaces to identify elements, properties and submodels. Khai et al. [10] proposed an approach for consistency checking of class and sequence diagrams based on Prolog. Cabot et al. [21] presented an automatic method for the verification of UML class diagrams extended with OCL constraints. They transformed the UML/OCL model into a Constraint Satisfaction Problem. The correctness properties such as weak and strong satisfiability or absence of constraint redundancies are checked. Khan and Porres [7] proposed an approach to automatically validate the consistency of UML models using logic reasoners for the Web Ontology Language OWL 2. They translated the models into OWL 2 and presented a tool supporting UML modeling tools to perform the translation from the models into the OWL 2.

In this paper, we propose an approach for the fully automatic, expressive verification of UML class diagrams and object diagrams using Prolog. First we formalize the model elements based on metamodel and convert the models into Prolog facts. Then we summarize several different types of consistency problems and encode into Prolog rules. Finally the

*Corresponding authors: hzbzhu@sei.ecnu.edu.cn (H. Zhu).
glu@cs.ecnu.edu.cn (G. Lu).

reasoning engines such as SWI-Prolog, are used to check the inconsistencies through analyzing the models and feedback the error information if any inconsistency error occurs. According to our approach, the design errors can be effectively avoided in design phase and the correctness of code-generalization can be guaranteed.

This paper is structured as follows. In Section II, we introduce the background about model consistency and our approach. Section III presents the formalization and conversion from UML models to Prolog. Section IV describes the different types of consistency. We have implemented a prototype tool which automatically translates the UML models into Prolog codes in Section V. Section VI shows some experiments to indicate the performance of our approach. Section VII concludes the paper and discusses the further work.

II. BACKGROUND

A. Consistency checking of UML models

In the past few years, the consistency problems in UML have become a hot issue. The term *model consistency* [18] is defined as “the overlapping elements in different models of the same system satisfy certain properties”. There are many studies that classify the consistency of models. One of the most widely accepted classifications is the Engels’ classification [6], which classifies the model consistency into four categories:

(1) *Vertical consistency* occurs when an abstract model is refined into a more concrete model. It is desirable that the concrete model should be consistent with the abstract one. The refinements of models cause the vertical consistency problems. (2) *Horizontal consistency* arises in case where different models describe the same system from different aspects containing overlapping elements. The overlapping elements should satisfy some elementary properties to ensure the consistency between the different models.

(3) *Syntactical consistency* ensures that a model conforms to the abstract syntax. The abstract syntax of models is usually defined by the metamodel. In other word, we consider that the models are syntactically consistent if the models are the instances of classes and associated by the instances of associations in the metamodel.

(4) *Semantic consistency* occurs when the developers expect to get more accurate models through additional constraints on models. The semantics of UML is usually specified in natural languages and OCL. It is hard to check the semantic consistency in UML especially for static diagrams since the OCL is not precisely defined in UML.

In this paper, we mainly consider three kinds of consistency issues: horizontal, syntactical and semantic consistency. The vertical consistency involving the refinements of the models is out of the scope of this paper.

B. Overview of the approach

We propose a general framework to check the consistency for a subset of UML static diagrams including class diagrams and object diagrams. The basic route of our approach is shown

as Fig. 1. First the designer provides a target model, created by UML CASE tools, and transforms the models into XMI files. Then the concepts of the models are automatically converted to the facts in Prolog database. Next the consistency rules with the facts are imported into the SWI-Prolog. The SWI-Prolog analyzes and queries the elements to verify the consistency of the target model and feedback the error information if any inconsistency occurs.

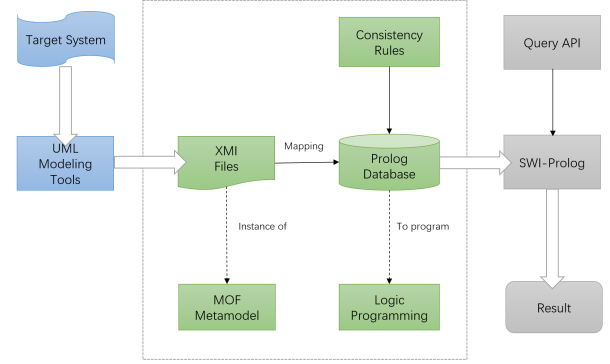


Fig. 1. An Outline of Our Approach

III. FROM UML MODELS TO PROLOG

A. From Class Diagrams to Prolog

The metamodel is a language that contains certain metadata describing the concepts and relations for providing a modeling language. The *Meta Object Facility* (MOF) standard defines the Essential MOF (EMOF), a subset of MOF that is used to define the metamodels. In this study, we formally define the syntax of the models according to the EMOF metamodel.

Definition 1: The syntax of the class diagrams is a structure

$$\mathcal{M} = \{class, attribute, operation, association, rolename, multiplicity, \prec\}.$$

where

- *class* is a set of classes.
- *attribute* is a set of signatures for functions mapping a class c to an associated attribute value.
- *operation* is a set of signatures for user-defined operations of a class c .
- *association* is a set of associations.
- *rolename* is a set of roles.
- *multiplicity* is a set of multiplicities of associations.
- \prec is a partial order on classes reflecting the generalization hierarchy of classes.

The class diagrams are denoted as a series of Prolog facts. Each model element in \mathcal{M} is described as a set of facts with the same clause and different parameters. Every element of class diagrams is assigned an identifier that identifies the actual objects that are to be instances of the various classes in the metamodel. In the following, each model elements is considered in details.

The most important part of the class diagrams is the classes. A class is a collection of objects that have the same attributes, operations, relationships, and semantics. The `class/3` clause in Prolog is used to denote the class, including an identifier, a class name and a boolean type indicating whether it is abstract.

class(classid, classname, isAbstract).

Note that the abstractions of the objects are classes, and the instantiations of the class are objects. The classes can be considered as the types of objects according to the Model Driven Architecture (MDA).

The classes define a group of objects' states and behaviors. More specifically, the attributes and associations of classes define the objects' states and relationships respectively, and the operations describe the behaviors of objects. The attributes are the values describing the object's properties, including a name and a type that specifies the domain of values. The `attribute/4` clause describes the attributes of class diagrams in Prolog.

attribute(attrid, attrname, attrtype, classid).

where `attrid` is an identifier of attribute, `attrname` is an attribute name, `attrtype` is the type of attribute, and `classid` is an identifier of class to which this attribute belongs.

The operations are parts of a class declaration in models. They describe the behavioral properties of classes, represented by the `operation/4` clause in Prolog.

operation(opid, opname, [parameters], classid).

where `opid` is an identifier of the operation, `opname` is a name of the operation, `[parameters]` is a list of parameters' id, and `classid` is an identifier of class to which this operation belongs.

The `parameter/3` clause in Prolog denotes the parameters in operations.

parameter(pid, pname, ptype).

where the `pid` and `pname` denote the identifier and name of the parameter respectively, and `ptype` denotes the type of the parameter, including the primitive types and class types.

The associations describe the structural relationships between the classes. In general, a class can have more than one associations, and an association can connect two or more classes. In this study, only binary associations are considered, the n -ary associations can be obtained by extending parameters in the `association/4` clause, where the `assoctype` can be directional, nondirectional, aggregate or compositive.

association(associd, classAid, classBid, assoctype).

In an association, the class can appear more than once playing different roles. The role names are usually useful in the navigations of models. The `rolename/4` clause assigns a unique role name to each class that participates in the binary association. The order of names in role names should coincide

with the order of classes in the corresponding associations. If the role names are omitted in a class diagram, we define the role names by changing the first letter of the name of target class to lower case.

rolename(roleid, nameA, nameB, associd).

Associations may also have multiplicities which specify the possible numbers of links for associated classes. The multiplicity describes the number of allowable objects of a range class to link with the objects of a domain class. The `multiplicity/5` clause has a minimum and maximum number of instances of the target classes, defined by the `lowval` and `upval` attributes. Unbounded ranges can be modelled using the value n for the upper attribute in Prolog.

multiplicity(multid, classid, lowval, upval, associd).

A generalization indicates that one of the two related class (subclass) is considered to be a specialized form of the other (superclass). The superclass is a generalization of the subclass. Generalization relationships form a hierarchy over the set of classes. A generalization hierarchy \prec is a partial order on the set of classes, shown as the `generalization/3` clause.

generalization(genid, subid, superid).

We define a recursive function `parents` to get all superclasses of a given class as follows.

$$\text{parents} : \begin{cases} \text{class} \rightarrow \mathcal{P}(\text{class}) \\ c \mapsto \{c' \mid c' \in \text{class} \wedge c \prec c'\} \end{cases} \quad (1)$$

The `parents/2` clause indicates the parent classes can be directly or indirectly derived through the generalization relationships. We can query the parent classes using `all_parents/2` where `findall` get all parent classes from the `parents` and put them into the variable `IDS`.

```
parents(Superid, Subid) :-
    generalization(_, Subid, Superid).
parents(Superid, Subid) :-
    generalization(_, Subid, X),
    parents(Superid, X).
all_parents(Subid, IDS) :-
    findall(Y, parents(Y, Subid), IDS).
```

B. From Object Diagrams to Prolog

The object diagrams show a complete or partial view of the structure of a modelled system at a specific time. The object diagrams is composed of the snapshots of running systems. An instance of a class is called an *object*, whereas an instance of an association is called a *link* that is a connection between two or more objects of classes at corresponding positions in the association. The object diagram focuses on the set of objects and attribute values, and the links between these objects at a particular time.

Definition 2: An object diagram for a model \mathcal{M} is a structure

$$\sigma(\mathcal{M}) = \{\text{object}, \text{link}, \text{attrval}\}.$$

where

- *object* is a set of objects.
- *link* is a set of links connecting objects.
- *attrval* is a set of functions assigning attribute values to each object.

The representation of the object diagrams in Prolog is similar to the representation of class diagrams.

```
object(objid,objname,classid).
link(linkid,objAid,objBid,associd).
attrval(attrvalid,attrid,value,objid).
```

The finite sets of *object*/3 clauses contain all objects and the finite sets of *link*/4 clauses contain links connecting objects. The *attrval*/4 clauses assign attribute values to each object. The *classid* in *object*/3 should be related to the *classid* in *class*/3, the same as *associd* and *attrid*.

Definition 3: The domain of a class is defined as a set of object identifiers that are the instances of the class.

$$\text{domain}(c) = \bigcup \{ \text{objects}(c') \mid c' \in \text{class} \wedge c' \prec c \}. \quad (2)$$

where the function *objects* gets all the objects of a given class. The domain of a class is defined using recursive predicate in Prolog. The *all_objects_ids*/2 returns the list of objects' identifiers for a given class identifier.

```
objects_ids(Classid, Objid) :-
    object(Objid, _, Classid).
objects_ids(Classid, Objid) :-
    generalization(_, Z, Classid),
    objects_ids(Z, Objid).
all_objects_ids(Classid, IDS) :-
    findall(Y, objects_ids(Classid, Y), IDS).
```

IV. CONSISTENCY CHECKING RULES

A. Syntactical Consistency

A metamodel defines the abstract syntax of the models. The models are syntactically consistent if they conform to the metamodel. More specifically, the elements of the model should be the instances of the classes in the associated metamodel, and the links of two elements are the instances of associations related by the associated classes in the metamodel. The MDA is described in four-layer architecture, each layer model can be regarded as instances of the upper layer model. The representation of UML concepts in this study is based on the EMOF metamodel. The models defined in our approach satisfy the syntactical consistency since they are the instances of the metamodel.

B. Semantic Consistency

A metamodel may also define a set of validity constraints on the metamodel using OCL, called semantic consistency [19]. For instance, a class should not define two attributes having same names. These OCL constraints are defined between the metamodels and models in order to describe more detailed

models. In a sense, any OCL expressions can be converted to the Prolog rules having the same semantics. The models are semantically consistent if the models conform to the rules. First the OCL representation is presented, and then we give the Prolog code for these constraints. Only several common constraints are presented because of the limited space.

Name Unique. The names of classes and associations should be unique and the names of attributes are unique in one class.

```
context Class inv:
    self.attribute -> forall(a1, a2 : attribute |
        a1 <> a2 implies a1.name <> a2.name).
```

Acyclic Generalization. There are no direct or indirect cycles in the generalization relationship.

```
context Class inv :
    self.allParents -> excludes(self).
```

The *list_reps*/2 shows the repetitions in the list. The list of names is the first parameter and the result of repetition elements is the second parameter. The circular generalization error occurs if any class is in the list of its parents classes. The *inheritSelf* returns true if the parameter class *X* has cycles in the generalization relationship.

```
list_reps([], []).
list_reps([X|Xs], Ds1) :-
    x_reps_others_fromlist(X, Ds, Os, Xs),
    list_reps(Os, Ds0),
    append(Ds, Ds0, Ds1).
x_reps_others_fromlist(_, [], [], []).
x_reps_others_fromlist(X, [X|Ds], Os, [X|Ys]) :-
    x_reps_others_fromlist(X, Ds, Os, Ys).
x_reps_others_fromlist(X, Ds, [Y|Os], [Y|Ys]) :-
    dif(Y, X),
    x_reps_others_fromlist(X, Ds, Os, Ys).
canGoTo(X, N, Nodes) :-
    member(X2, [X|Nodes]),
    generalization(_, X2, X1),
    \+ member(X1, Nodes),
    canGoTo(X, N, [X1|Nodes]).
canGoTo(_, N, N).
canGoTo(X, Nodes) :-
    canGoTo(X, Nodes, []).
inheritSelf(X) :-
    canGoTo(X, Nodes), member(X, Nodes), !.
```

We also offer query interfaces to query properties in SWI-Prolog. For example, the variable *List* indicates the set of the classes' name and the variable *R* represents the repetition elements if there are any repeating elements in the models.

```
?- bagof(Y, X^class(X, Y, _), List), list_reps(List, R).
```

C. Horizontal Consistency

The problems of horizontal consistency is caused by the overlapping elements of different diagrams. There are some overlapping elements between the class diagrams and object diagrams. For instance, a class should exist in the class diagram if the objects of the class exist in the object diagram.

When the inconsistency occurs, we can quickly locate the wrong place based on the feedback information.

Class Existence. The related classes of objects in the object diagrams should exist in the class diagrams since the objects are the instances of the classes.

$$\forall o \in \text{object}, \exists c \in \text{class}, o.\text{classid} = c.\text{classid}.$$

Association Existence. There are links between objects in object diagrams and the associations referred to these links between corresponding classes exist in class diagrams.

$$\begin{aligned} \forall l \in \text{link}, \exists o_1, o_2 \in \text{object}, \exists a \in \text{association}, \\ l.\text{objAid} = o_1.\text{objid} \wedge l.\text{objBid} = o_2.\text{objid} \wedge \\ o_1.\text{classid} = a.\text{classAid} \wedge o_2.\text{classid} = a.\text{classBid} \wedge \\ a.\text{associd} = l.\text{associd}. \end{aligned}$$

Generalization Satisfaction. If class c_1 is a sub class of class c_2 , the domain of c_1 is a subset of the domain of c_2 . The set of objects connected to its subclasses should also be disjoint.

$$\forall c_1, c_2 \in \text{class}, c_1 \prec c_2 \Rightarrow \text{domain}(c_1) \subseteq \text{domain}(c_2).$$

$$\forall c \in \text{class}, \text{domain}(c) = \bigcup_{c_i \in \text{sub}(c)} \text{domain}(c_i).$$

$$\forall c \in \text{class}, \bigcap_{c_i \in \text{sub}(c)} \text{domain}(c_i) = \emptyset.$$

where function $\text{sub}(c)$ gets all the subclasses of class c . Note that the domain of an abstract class is comprised of the domain of the subclasses since there are no instances of an abstract class.

Multiplicity Satisfaction. The number of the instances of an association must satisfy the multiplicity. A set of links should satisfy the multiplicity specifications defined for an association. A minimum and maximum number of instances of target classes must be satisfied using the *lower* and *upper* functions.

$$\forall as \in \text{association}, \text{lower}(\text{multiplicity}(as)) \leq \text{card}\{l' \in \text{link}(as)\} \leq \text{upper}(\text{multiplicity}(as)).$$

Part of the representation of the horizontal consistency rules in Prolog is listed as follows:

```
object_rule(Objectid, Classid) :-
    object(Objectid, _, Classid),
    class(Classid, _, _),
    write("Object: "), write(Objectid), nl,
    write("Class: "), write(Classid).
assoc_exist(Msgid, Sndobjid, Recobjid,
            ClassA, ClassB) :-
    link(Msgid, Sndobjid, Recobjid),
    object(Sndobjid, _, ClassA),
    object(Recobjid, _, ClassB),
    association(_, ClassA, ClassB);
    association(_, ClassB, ClassA).
gen_rule(Superid, Subid) :-
    generalization(_, Superid, Subid),
    all_objects_ids(Subid, IDS), write(IDS), nl,
    all_objects_ids(Superid, IDS2), write(IDS2), nl,
    subset(IDS, IDS2).
```

V. REASONING

We provide an automated tool to transform the models into Prolog. The original models are described in XMI format, generated directly from the UML CASE tools such as StarUML and Astah. Then the models in XMI format are transformed into the Prolog facts automatically. The algorithm of automatic transformation is shown as follows. The code is also available online in [22].

Algorithm 1 Transformation From class diagrams To Prolog

Require: The models represent in XMI format.

Ensure: The Prolog Facts of the model.

```
1: for each packagedElement ∈ uml:model do
2:   exact(Class.name).
3:   exact(Association.name).
4: end for
5: for each ownedAttribute ∈ uml:Class do
6:   if hasAttribute('association') then
7:     exact(association detail).
8:     exact(multiplicity).
9:   else
10:    exact(attribute).
11:   end if
12: end for
13: for each association ∈ association list do
14:   exact(rolename).
15: end for
16: for each ownedOperation ∈ uml:Class do
17:   exact(operation).
18: end for
19: for each generalization ∈ uml:model do
20:   exact(generalization).
21: end for
22: return The Prolog facts of the model.
```

After the model conversion to Prolog facts, the consistency rules along with the converted models are import into SWI-Prolog. The SWI-Prolog checks whether the Prolog-based models satisfy the consistency rules. The models are consistent if the models satisfy all consistency rules. If there are any models that cannot satisfy rules, the SWI-Prolog gives the error messages. Besides, the query interfaces, similar to OCL, are provided to query the relevant information in the models.

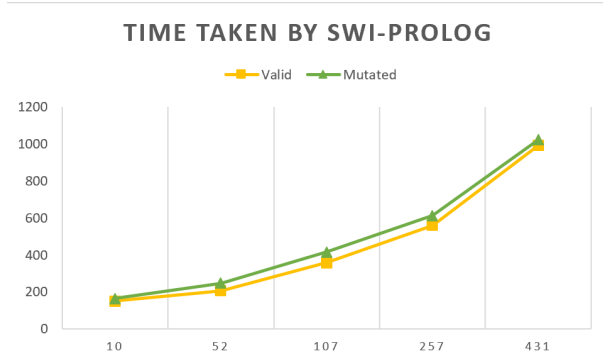
VI. PERFORMANCE EVALUATION

In order to determine the performance of the translation and reasoning tools, we conducted an experiments using UML class diagrams and object diagrams with invariants consisting of 10 - 437 model elements. We use a desktop computer with an Intel(R) Core(TM) i5-7400 CPU processor running at 3.00GHz with 16GB of RAM. The results are shown in Table I.

The performance tests are conducted for both consistent and mutated models. The mutated models contain the randomly introduced inconsistencies such as the same names of models, the direct and indirect cycles in generalizations and

TABLE I
TIME TAKEN BY THE TRANSITION TOOL AND REASONING ENGINES TO
PROCESS UML MODELS

Classes	4	7	16	19	32
Model Element	10	52	107	257	431
Translation time (ms)	9.3	39.1	66.7	140.2	321.3
Valid (ms)	150.2	206.9	359.0	561.3	993.5
Mutated (ms)	165.3	248.1	415.3	613.2	1023.1



the multiplicity inconsistencies between the class diagrams and object diagrams. For each test, we measure the time required to translate a model from UML to Prolog and the time required by the SWI-Prolog reasoner to analyze the models. The experimental results show that our approach can find all the inconsistencies mentioned in this paper and the time complexity of Prolog reasoning the consistencies is linear.

VII. CONCLUSION AND FURTHER WORK

In this paper, we present a Prolog-based consistency checking for UML class diagrams and object diagrams. We have implemented an automatic transformation from the models in XMI format to the Prolog facts. Then the SWI-Prolog is used to check whether the facts satisfy the consistency rules. The models are consistent if all consistency rules are satisfied. The reasoning engine will give error information if any inconsistency occurs. Besides, we provide query interfaces, similar with the OCL, to query the relevant information. Our work provides a novel approach to automatically detect the consistency of models and promise the errors will not propagate to the implementation stage.

This study only gives a brief formalization of class diagrams and object diagrams. However, these diagrams are not enough to describe the whole system in real world. We expect to construct a complete UML framework that covers the static structure and dynamic behavior of the systems. Besides, the types of consistency checking in this study are far from enough. More consistency checking rules should be covered to have a confidence for systems.

VIII. ACKNOWLEDGEMENT

This work was partly supported by National Natural Science Foundation of China (Grant No. 61872145) and Shanghai

Collaborative Innovation Center of Trustworthy Software for Internet of Things (No.ZF1213).

REFERENCES

- [1] G. Booch, The Unified Modeling Language User Guide, Pearson Education India, 2005.
- [2] R. Van Der Straeten, J. Simmonds, and T. Mens, Detecting Inconsistencies between UML Models Using Description Logic, *Description Logic*, vol. 81, 2003.
- [3] J. Woodcock and J. Davies, Using Z: specification, refinement, and proof, Prentice Hall Englewood Cliffs, vol. 39, 1996.
- [4] C. Snook and M. Butler, UML-B: Formal modeling and design aided by UML, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15(1), pp. 92–122, 2006.
- [5] R. S. Bashir, S.P. Lee, S.U.R. Khan, V. Chang, and S. Farid, UML models consistency management: Guidelines for software quality manager, *International Journal of Information Management*, vol. 36(6), pp. 883–899, 2016.
- [6] G. Engels, J.M. Küster, R. Heckel, and L. Groenewegen, A methodology for specifying and analyzing consistency of object-oriented behavioral models, *ACM SIGSOFT software engineering notes*, vol. 26(5), pp. 186–195, 2001.
- [7] A. H. Khan and I. Porres, Consistency of UML class, object and statechart diagrams using ontology reasoners, *Journal of Visual Languages & Computing*, vol. 26, pp. 42–65, 2015.
- [8] M. Soeken, R. Wille, and R. Drechsler, Verifying dynamic aspects of UML models, *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, 2011.
- [9] J. Chimiak-Opoka, M. Felderer, and C. Lenz and C. Lange, Querying UML models using OCL and Prolog: A performance study, *IEEE International Conference on Software Testing Verification and Validation Workshop*, pp. 81–88, 2008.
- [10] Z. Khai, A. Nadeem, and G. Lee, A Prolog Based Approach to Consistency Checking of UML Class and Sequence Diagrams, *International Conference on Advanced Software Engineering and Its Applications*, pp. 85–96, 2011.
- [11] M. Egea and V. Rusu, Formal executable semantics for conformance in the MDE framework, *Innovations in Systems and Software Engineering*, vol. 6(1-2), pp. 73–81, 2010.
- [12] H. Störle, A Prolog-based Approach to Representing and Querying Software Engineering Models, *VLL*, vol. 274, pp. 71–83, 2007.
- [13] P. Krishnan, Consistency checks for UML, *7th Asia-Pacific Proceedings on Software Engineering*, pp. 162–169, 2000.
- [14] A. Tsiolakis, Consistency analysis of UML class and sequence diagrams based on attributed typed graphs and their transformation, *ETAPS 2000 workshop on graph transformation systems*, 2000.
- [15] L. C. Briand, Y. Labiche, L. Osullivan, and M.M. Sowka, Automated impact analysis of UML models, *Journal of Systems and Software*, vol. 79(3), pp. 339–352, 2006.
- [16] D. Torre, Y. Labiche, and M. Genero, UML consistency rules: a systematic mapping study, *18th International Conference on Evaluation and Assessment in Software Engineering*, pp. 6, 2014.
- [17] D. Torre, Verifying the consistency of UML models, *2016 IEEE International Symposium on Software Reliability Engineering Workshops*, pp. 53–54, 2016.
- [18] G. Spanoudakis and A. Zisman, Inconsistency management in software engineering: Survey and open research issues, *Handbook of software engineering and knowledge engineering*, vol. 1, pp. 329–380, 2001.
- [19] X. Thirioux, B. Combemale, X. Crégut, and P. Garoche, A framework to formalise the MDE foundations, *International Workshop on Towers of Models*, pp. 14–30, 2007.
- [20] A. Andres and H.D. Rombach, A handbook of software and systems engineering: Empirical observations, laws, and theories, Pearson Education, 2003.
- [21] J. Cabot, R. Clarisó, and D. Riera, On the verification of UML/OCL class diagrams using constraint programming, *Journal of Systems and Software*, vol. 93, pp. 1–23, 2014.
- [22] F. Sheng, Transformation from UML to Prolog. “<https://github.com/shengfeng/xmi2pl>”, 2018

Modeling and Verifying TESAC Using CSP

Dongzhen Sun¹, Huibiao Zhu^{*1}, Yuan Fei^{*2}, Lili Xiao¹, Gang Lu¹, Jiaqi Yin¹

¹Shanghai Key Laboratory of Trustworthy Computing,
School of Computer Science and Software Engineering
East China Normal University, Shanghai, China

²School of Information, Mechanical and Electrical Engineering,
Shanghai Normal University, Shanghai, China

Abstract—Cloud computing is an emerging computing paradigm in IT industries. The wide adoption of cloud computing is raising concerns about management of data in the cloud. Access control and security are two critical issues of cloud computing. Time efficient secure access control (TESAC) model is a new data access control scheme which can minimise many significant problems. This scheme has better performance than other existing models in a cloud computing environment. TESAC is attracting more and more attentions from industries. Hence, the reliability of TESAC becomes extremely important. In this paper, we apply Communication Sequential Processes (CSP) to model TESAC, as well as their security properties. We mainly focus on its data access mechanism part and formalize it in detail. Moreover, using the model checker Process Analysis Toolkit (PAT), we have verified that the TESAC model cannot assure the security of data with malicious users. For the purpose of solving this problem we introduce a new method similar to digital signature. Our study can improve the security and robustness of the TESAC model.

Index Terms—TESAC; Cloud computing; CSP; Access control; Modeling; Verification;

I. INTRODUCTION

Cloud computing is considered to be an important driver of world-wide IT industries [1]. With the development of cloud computing technology, many data access models (ACMs) have been proposed by researchers [2]–[4]. Time efficient secure access control (TESAC) [5] model is a new data access control scheme which uses asymmetric encryption to guarantee data security. This scheme is more efficient than other existing solutions after an evaluation in terms of both theoretical and experimental results.

The main types of access control models can be roughly divided into three categories [6]: Mandatory Access Control (MAC), Discretionary Access Control (DAC), Role Based Access Control (RBAC). In order to achieve the goal of the network as a high-performance computer, cloud service providers specify suitable access control policies for users to access data and other resources. For limiting user access rights in different situations, Ferraiolo et al. [2] put forward the role-based access control (RBAC), where cloud service provider determines the user access to the system by means of the job role. Gao et al. [3] came up with novel data access control (NDAC) to ensure secured and confidential data communication between users and cloud servers. For the express purpose of providing a superior decision-making ability, Danwei et al. [4] presented usage control-based access (UCON) model which can easily implement the security strategy of DAC, MAC and RBAC. It is a scheme that combines all the merits of the traditional access control models. These models attach great importance to security issues, but they adopt only informal methods to analyze and there is few work on the formalization of these

models. Due to the superb characteristics of TESAC, the industry will definitely draw a great amount of interests in this scheme. So it is extremely important to verify and analyze it by formal methods. we use classical process algebra language Communicating Sequential Processes (CSP) [7], [8] to model TESAC, and in terms of a model checking tool Process Analysis Toolkit (PAT) [9] to verify some vital properties. The verification results demonstrate and show that the security problem really exists in this scheme. Through the formalization, we want to provide a deeper understanding of TESAC as well as its security properties, and we hope that other researchers can realize the security problems that may presence in their system from our verification results.

This paper is organized as follows: Section II presents an overview of TESAC, as well as introduction to CSP. Section III is devoted to the modelling of TESAC. The results that we use PAT to verify the original model are presented in Section IV, and give the improvement for better performance. Finally, conclusions and future directions are given in Section V.

II. BACKGROUND

In this section, we will give a briefly introduce of TESAC, especially its data access control scheme in cloud computing. After that, we also give a brief introduce of process algebra CSP.

A. TESAC

TESAC is a new data access control scheme based on the users' profiles. It is different from traditional access control models implementing a cloud environment. It can resolve many problems, such as high overhead of the system, high searching time for providing the public key of the data owner, high data accessing time, etc. The scheme consists of three entities:

- **Cloud service provider:** It possesses a number of servers having sufficient storage space and power to provide infrastructure and cloud services for both users and data owners.
- **Data owner:** It sends its encrypted data or file to the cloud database and managed by servers. Data owner can be any user.
- **User:** It must be registered at the cloud server for accessing a data or file.

The specific meaning of some notations of keys are listed in Table I. The process of encryption and decryption is given in Steps 1-8. When the user wants to get data from server, the following sequence of actions occurs:

- 1) First, the user sends a data access request to the server.
- 2) When the server receives the request, it encrypts PUOWN by using its own PRSP and PUUSR, and then, provides it to the users.
- 3) The user obtains PUOWN through two layers of decryption. Then the user sends a request to the data owner for obtaining the secret key and the certificate. The user encrypts the request message with PRUSR and PUOWN, and then, sends it to the data owner.

Corresponding authors: hbzhu@sei.ecnu.edu.cn (H. Zhu),
yuanfei@shnu.edu.cn (Y. Fei).

- 4) The data owner uses PROWN and PUUSR to decrypt the package to get the users message. The data owner needs to verify the legality of the user to the server.
- 5) The server sends a feedback message to the data owner.
- 6) If it is a positive feedback, the data owner uses PROWN and PUUSR to encrypt its secret and certificate. Finally, the data owner sends this package to the user.
- 7) The user uses PRUSR and PUOWN to decrypt the message. After the user acquires the secret and the certificate, it uses PRUSR and PUSP to encrypt the certificate. The user then sends it to the server.
- 8) The server decrypts the message. If the user's presented certificate is matched with the corresponding certificate of the requested data. The server provides the data to the user. Finally, the user uses PRUSR, PUSP and secret to decrypt the message to get the desired data.

TABLE I
NOTATION AND DESCRIPTION

PUOWN	public key of the data owner
PROWN	private key of the data owner
PUUSR	public key of the user
PRUSR	private key of the user
PUSP	public key of the server
PRSP	private key of the server

B. A brief overview of CSP

In this subsection we give a brief overview of CSP (Communication Sequential Processes). It is a process algebra proposed by Hoare in 1978. As one of the most mature formal methods, it is tailored for describing the interaction between concurrency systems by mathematical theories. Because of its well-known expressive ability, CSP has been widely used in many fields [10]–[12].

CSP processes are constituted by primitive processes and actions. We use the following syntax to define the processes in this paper, whereby P and Q represent processes, the alphabets $\alpha(P)$ and $\alpha(Q)$ mean the set of actions that the processes P and Q can take respectively, and a and b denote the atomic actions and c stands for the name of a channel. The syntax of CSP is given as below.

$$P, Q ::= \text{Skip} \mid \text{Stop} \mid a \rightarrow P \mid c!v \rightarrow P \mid c?x \rightarrow P \mid P; Q \\ \mid P \parallel Q \mid P \square Q \mid P \triangleleft b \triangleright Q \mid b * Q \mid P[[a \leftarrow b]]$$

- 1) **Skip** indicates a basic process that terminates successfully.
- 2) **Stop** represents that a process can't do anything any more.
- 3) $a \rightarrow P$ denotes a process first engages in action a , then acts due to the specification of process P .
- 4) $c!v \rightarrow P$ describes a process sends a value v through channel c , then behaves according to the specification of P .
- 5) $c?x \rightarrow P$ represents a process receives a value and assigns it to the variable x , then the behavior is like process P .
- 6) $P; Q$ indicates only when process P has terminated can process Q start to perform.
- 7) $P \parallel Q$ means process P and process Q perform in parallel. And they must synchronize facing the same events.
- 8) $P \square Q$ stands for external choice. A process behaves following the specification of process P or Q . However, the choice depends on the environment.
- 9) $b * Q$ expresses circulation. If the value of variable b is true, a process behaves like process Q circularly. Otherwise, it ends the circulation.
- 10) $P \triangleleft b \triangleright Q$ shows conditional choice. If the condition b is true, a process acts like P , otherwise, like Q .
- 11) $P[[a \leftarrow d]]$ indicates event a is replaced by d in process P .

III. MODELING TESAC

A. Sets, Messages and Channels

In order to formalize the protocol more conveniently, we give the fundamental information about sets, messages and channels. We define seven sets in our model. **Entity** set represents entities including servers, users and data owners. **Req** set denotes request and confirming messages. **ACK** set means feedback messages. **Content** set contains the content to be encrypted. **PUKey** set contains all the public key of entities, **PRKey** set contains all the private key of entities, **Sec** set contains the key that the data owner uses to encrypt the data.

Two core elements of modeling are internal processing procedures of entities and message packets transmitted between entities. Based on the sets defined above, we abstract them into different messages. We use the form $E(k, d)$ to indicate that k is utilized to encrypt the message d . Each message includes a tag from the set $\{msg_{req}, msg_{key1}, msg_{data1}, msg_{ack}, msg_{key2}, msg_{data2}, msg_{ackin}\}$.

The messages that are transmitted among entities as follows:

$$\begin{aligned} MSG_{req} &= \{msg_{req}.a.b.req, msg_{req}.a.b.cof \mid \\ &\quad a, b \in Entity, req, cof \in Req\} \\ MSG_{key1} &= \{msg_{key1}.a.b.E(K_1, E(K_2^{-1}, d)) \mid a, b \in Entity, \\ &\quad K_1 \in PUKey, K_2^{-1} \in PRKey, d \in Content\} \\ MSG_{data1} &= \{msg_{data1}.a.b.E(K_1, E(K_2^{-1}, E(K, d))) \mid \\ &\quad a, b \in Entity, K_1 \in PUKey, K_2^{-1} \in PRKey, \\ &\quad K \in Sec, d \in Content\} \\ MSG_{ack} &= \{msg_{ack}.a.b.x \mid a, b \in Entity, x \in Ack\} \\ MSG_{key2} &= \{msg_{key2}.E(K_1, (K_2^{-1}, d)).K_1^{-1}.K_2 \mid d \in Content, \\ &\quad K_1, K_2 \in PUKey, K_1^{-1}, K_2^{-1} \in PRKey\} \\ MSG_{data2} &= \{msg_{data2}.E(K_1, E(K_2^{-1}, E(K, d))).K_1^{-1}.K_2.K \mid \\ &\quad K_1, K_2 \in PUKey, K_1, K_2 \in PUKey, \\ &\quad K_1^{-1}, K_2^{-1} \in PRKey, K \in Sec, d \in Content\} \\ MSG_{ackin} &= \{msg_{cont}.y \mid y \in Content\} \\ MSG_{out} &= MSG_{req} \cup MSG_{key1} \cup MSG_{data1} \cup MSG_{ack} \\ MSG_{in} &= MSG_{key2} \cup MSG_{data2} \cup MSG_{ackin} \\ MSG &= MSG_{out} \cup MSG_{in} \end{aligned}$$

MSG_{req} represents the set of request messages. MSG_{key1} stands for the set of two-layer encryption messages. MSG_{data1} indicates the set of messages whose real data encrypted by public and private keys. MSG_{ack} denotes feedback messages. MSG_{key2} stands for the set of three-layer encryption messages. MSG_{data2} indicates messages sent to the process specially for internal processing. MSG_{ackin} represents messages that return data to entities by internal process. MSG_{out} represents the set of messages transmitted between entities, MSG_{in} denotes internal processing messages of entities.

Then, we give the definitions of channels to model the communications between processes:

- channels between users, data owners and servers, denoted by COM_PATH : ComUS, ComUD, ComDS.
- channels of intruders who intercept users, data owners and servers, represented by $INTRUDER_PATH$: FakeU, FakeS, FakeD.
- channels of processing messages, depicted by $PROCESS_PATH$: GetU, GetD, GetS.

The declarations of the channels are as follows:

$$\begin{aligned} \text{Channel } COM_PATH, INTRUDER_PATH &: MSG_{out} \\ \text{Channel } PROCESS_PATH &: MSG_{in} \end{aligned}$$

B. Overall Modeling

As mentioned above, the whole scheme contains three important entities, including *User*, *DataOwner* and *Server*. We formalize the whole system as below.

$$\begin{aligned} \text{System} &=_{df} \text{System}_0[|INTRUDERPATH|] \text{Intruder} \\ \text{System}_0 &=_{df} \text{User}[|DataOwner|] \text{Server} \\ \text{User} &=_{df} \text{User}_1[|PROCESSPATH|] \text{ProcessU} \\ \text{DataOwner} &=_{df} \text{DataOwner}_1[|PROCESSPATH|] \text{ProcessD} \\ \text{Server} &=_{df} \text{Server}_1[|PROCESSPATH|] \text{ProcessU} \end{aligned}$$

User, *DataOwner* and *Server*, as their names demonstrate, represent the user, the data owner and the server. *ProcessU*, *ProcessD* and *ProcessS* denote the internal processing procedure of the user, the data owner and the server. Considering the existence of intruders, we also build process *Intruder* to simulate the behavior of intruders who eavesdrop and modifies messages. Interprocess communication between processes are illustrated in Fig.2.

C. User Modeling

We first formalize process *User₀* to describe the behavior of a user process without intruders.

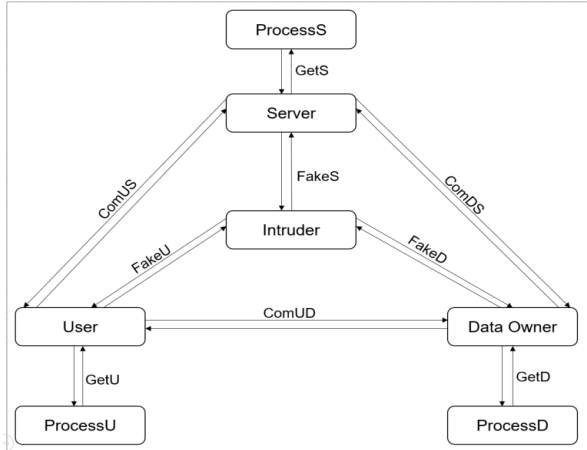
$$\begin{aligned} \text{User}_0 &=_{df} \\ &\text{ComUS!msgreq}.U.S.reqdata \rightarrow \\ &\text{ComUS?msgkey1}.S.U.E(\text{PUUSR}, E(\text{PRSP}, \text{PUOWN})) \rightarrow \\ &\text{GetU!msgkey2}.E(\text{PUUSR}, E(\text{PRSP}, \text{PUOWN})). \\ &\text{PRUSR.PUSP} \rightarrow \text{GetU?msgackin}.PUOWN \rightarrow \\ &\text{ComUD!msgkey1}.U.D.E(\text{PUOWN}, E(\text{PRUSR}, reqsc)) \rightarrow \\ &\text{ComUD?msgkey1}.D.U.E(\text{PUUSR}, E(\text{PROWN}, (s, c))) \rightarrow \\ &\text{GetU!msgkey2}.E(\text{PUUSR}, E(\text{PROWN}, (s, c))). \\ &\text{PRUSR.PUOWN} \rightarrow \text{GetU?msgackin}.s.c \rightarrow \\ &\text{ComUS!msgkey1}.U.S.E(\text{PUSP}, E(\text{PRUSR}, c)) \rightarrow \\ &\text{ComUS?msgdata1}.S.U.E(\text{PUUSR}, E(\text{PRSP}, E(s, data))) \rightarrow \\ &\text{GetU!msgdata2}. \\ &E(\text{PUUSR}, E(\text{PRSP}, E(s, data))).\text{PRUSR.PUSP}.s \rightarrow \\ &\text{GetU?msgackin}.data \rightarrow \text{User}_0; \end{aligned}$$


Fig. 1. Interprocess communication between processes in model

where, *req_data* represents the data request sent from the user to the server. *req_sc* represents the request of the secret and the certificate that the user sends to the data owner. *s* represents the secret and *c* represents the certificate. The six actions on channel *ComUS* and *ComUD* correspond to Steps 1-8 of User in Fig.1

in order. After receiving the encrypted messages, entity *User₀* sends them to internal processing part *ProcessU* by way of channel *GetU*, as well as accepts the decrypted messages.

Then the existence of intruder actions needs to take into consideration. For example, we must allow the instances of data request to be faked, the instances of responses of *PUOWN* to be intercepted, etc. We do this via renaming.

$$\begin{aligned} \text{User}_1 &=_{df} \text{User}_0[|ComUS?\{ComUS\} \leftarrow ComUS?\{ComUS\}|, \\ &ComUS?\{ComUS\} \leftarrow FakeU?\{ComUS\}|, \\ &ComUS!\{ComUS\} \leftarrow ComUS!\{ComUS\}|, \\ &ComUS!\{ComUS\} \leftarrow FakeS!\{ComUS\}|, \\ &ComUD?\{ComUD\} \leftarrow ComUD?\{ComUD\}|, \\ &ComUD?\{ComUD\} \leftarrow FakeU?\{ComUD\}|, \\ &ComUD!\{ComUD\} \leftarrow ComUD!\{ComUD\}|, \\ &ComUD!\{ComUD\} \leftarrow FakeD!\{ComUD\}|] \end{aligned}$$

$\{c\}$ denotes the set of all communications over channel *c*. Whenever *User₀* does an action on channel *ComUS* or *ComUD*, *User₁* will does a corresponding action on channels with prefix *Com* or channels with prefix *Fake*. Here, channels with prefix *Com* only include *ComUS* and *ComUD* and channels with prefix *Fake* only include *FakeU* and *FakeS*.

We can define CSP processes representing the data owner and server similarly.

D. ProcessU Modeling

In order to simulate the internal process of the user, we use *ProcessU* to deal with decrypting message. We must consider the possibility of intruder actions.

$$\begin{aligned} \text{ProcessU} &=_{df} \\ &\text{GetU?msgkey2}.E(\text{PUUSR}, E(\text{PRSP}, \text{PUOWN})). \\ &\text{PRUSR.PUSP} \rightarrow \\ &\left(\begin{aligned} &\text{GetU!msgackin}.PUOWN \rightarrow \text{ProcessU} \\ &\triangleleft(((\text{PUSP} == \text{PRSP})) \vee (\text{PUI} == \text{PRI})) \\ &\wedge (\text{PUUSR} == \text{PRUSR})) \triangleright \\ &(\text{GetU!msgackin}.NO \rightarrow \text{ProcessU}) \end{aligned} \right) \\ &\square \text{GetU?msgkey2}.E(\text{PUUSR}, E(\text{PROWN}, (s, c))). \\ &\text{PRUSR.PUOWN} \rightarrow \\ &\left(\begin{aligned} &\text{GetU!msgackin}.PUOWN \rightarrow \text{ProcessU} \\ &\triangleleft(((\text{PUOWN} == \text{PROWN})) \vee (\text{PUI} == \text{PRI})) \\ &\wedge (\text{PUUSR} == \text{PRUSR})) \triangleright \\ &(\text{GetU!msgackin}.NO \rightarrow \text{ProcessU}) \end{aligned} \right) \\ &\square \text{GetU?msgdata2}.E(\text{PUUSR}, E(\text{PRSP}, E(sec, data))). \\ &\text{PRUSR.PUSP}.s \rightarrow \\ &\left(\begin{aligned} &\text{GetU!msgackin}.PUOWN \rightarrow \text{ProcessU} \\ &\triangleleft(((\text{PUSP} == \text{PRSP})) \vee (\text{PUI} == \text{PRI})) \\ &\wedge (\text{PUUSR} == \text{PRUSR}) \wedge (sec == s)) \triangleright \\ &(\text{GetU!msgackin}.NO \rightarrow \text{ProcessU}) \end{aligned} \right) \end{aligned}$$

We use *PUI* to represent the public key of the intruder, *PRI* represents the private key of the intruder. *ProcessU* receives the encrypted message with decryption keys by channel *GetU*. Then it judges whether the decryption key can decrypt the message successfully. If the key does not match, *ProcessU* returns a negative message to *User₀*. Else, it sends the decrypted content to *User₀*. The internal process *ProcessD* of the data owner and *ProcessS* of the server can be defined similarly.

E. Intruder Modeling

Finally, we give the formalization of the intruder. We also regard the intruder as a process that can perform any attack as a real world intruder can be. It can intercept or fake messages in the

communication on channel *ComUS*, *ComUD* and *ComDS*. We define the set of facts that an intruder might learn as follows:

$$\begin{aligned} Fact =_{df} & \{U, S, D\} \cup \{PUOWN, PUUSR, PUSP\} \\ & \cup \{E(k, content) | k \in \{Sec, PUKey, PRKey\}, \\ & content \in Content, (s, c)\} \cup MSG_{out} \cup \{PUI, PRI\} \end{aligned}$$

Intruder can derive new facts from the set of *Facts* it has learned. We use the symbol $F \mapsto f$ to indicate that the fact f can be derived from the set F of facts. The definition is given as follows:

$$\begin{aligned} \{K_1^{-1}, E(K_1, E(K_2^{-1}, d))\} & \mapsto E(K_2^{-1}, d), \{K_2, E(K_2, d)\} \mapsto d \\ \{K_2, d\} & \mapsto E(K_2^{-1}, d), \{K_1, E(K_2^{-1}, d)\} \mapsto E(K_1, E(K_2^{-1}, d)) \\ F \mapsto f \wedge F \subseteq F' & \Rightarrow F' \mapsto f \end{aligned}$$

The first two rules represent encryption and the third and the fourth represent decryption. The final rule means if the intruder can derive the fact f from a set of facts F , then f can also be derived from a larger set F' .

We give a definition of **Info** function in which the intruders can learn by seeing the intercepted messages, shown as follows:

$$\begin{aligned} Info(msg_{req}.a.b.req) &=_{df} \{a, b, req\} \\ Info(msg_{req}.a.b.cof) &=_{df} \{a, b, cof\} \\ Info(msg_{key1}.a.b.E(K_1, (K_2^{-1}, d))) &=_{df} \{a, b, E(K_1, (K_2^{-1}, d))\} \\ Info(msg_{data1}.a.b.E(K_1, E(K_2^{-1}, E(k, d)))) &=_{df} \\ & \{a, b, E(K_1, E(K_2^{-1}, E(k, d)))\} \\ Info(msg_{ack1}.a.b.x) &=_{df} \{a, b, x\} \end{aligned}$$

where $a, b \in Entity$, $req, cof \in Req$, $K_1 \in PUKey$, $K_2^{-1} \in PRKey$, $d \in content$, $k \in Sec$, $x \in Ack$.

We define a channel *deduce* to be used for deducing new facts. The definition is given as follows:

Channel deduce: Fact.P(Fact)

All the messages transmitted between entities can be overheard by the intruder. It can deduce a new fact from ones it has already known.

It can also fake some messages if he knows all the sub-messages. The formalization of *Intruder₀* is defined as below:

$$\begin{aligned} Intruder_0(F) &=_{df} \\ \Box \Box m \in MSG_{out} FakeU?m \rightarrow \\ FakeS!m\{user_fake_success = true\} \rightarrow \\ Intruder_0(F \cup Info(m)) \\ \Box \Box m \in MSG_{out} FakeS?m \rightarrow \\ FakeU!m\{server_fake_success = true\} \rightarrow \\ Intruder_0(F \cup Info(m)) \\ \Box \Box m \in MSG_{out} FakeU?m \rightarrow \\ FakeD!m\{user_fake_success = true\} \rightarrow \\ Intruder_0(F \cup Info(m)) \\ \Box \Box m \in MSG_{out} FakeD?m \rightarrow FakeU!m \rightarrow \\ Intruder_0(F \cup Info(m)) \\ \Box \Box m \in MSG_{out} FakeD?m \rightarrow FakeS!m \rightarrow \\ Intruder_0(F \cup Info(m)) \\ \Box \Box m \in MSG_{out} FakeS?m \rightarrow \\ FakeD!m\{server_fake_success = true\} \rightarrow \\ Intruder_0(F \cup Info(m)) \\ \Box \Box f \in Fact, f \notin F, F \mapsto f deduce.f.F \rightarrow \\ Intruder_0(F \cup f) \end{aligned}$$

When *Intruder₀* receives some messages, if it is not encrypted by another entity with its public key, *Intruder₀* can replace some content in the message and send it to the original receiving entity; If the message is encrypted with the entity's public key, *Intruder₀* can not know or replace the content of the message. Because it does not know the private key that matches the public key. However, *Intruder₀* has its own public key and private key, allowing it to fake messages to send to the entities. We give the definition of *IK* to represent the initial knowledge of the intruder:

$$\begin{aligned} Intruder &=_{df} Intruder_0(IK) \\ IK &=_{df} \{U, S, D, PUI, PRI\} \end{aligned}$$

IV. VERIFICATION AND IMPROVEMENT

In this section, we will verify the four properties (deadlock freedom, user faking, server faking and protocol completeness) by virtue of the model checker PAT. According to the verification results, we improve the original model for a better safety performance.

A. Security Specification

We allow intruders to perform a series of intrusive actions and give some facts that intruders can learn. Under these conditions, whether the intruders can attack the system successfully by intercepting and faking. If the intruder can get *PUOWN* or data from the server, then it can be concluded that the system was successfully attacked by the intruder. This is in the ideal situation, but the protocol is running in an open environment, then *PUOWN* is public as a public key. Once an intruder knows the data owner's public key, it is critical that the system can detect and prevent the disclosure of the message. Thus we will verify our systems against the following specification:

$$\begin{aligned} SPEC_u &=_{df} CHAOS - \left(\sum -\{FakeU\} \right) \\ SPEC_d &=_{df} CHAOS - \left(\sum -\{FakeD\} \right) \\ SPEC_s &=_{df} CHAOS - \left(\sum -\{FakeS\} \right) \end{aligned}$$

If the system with intruders refines these specifications, then it would indeed be secure.

B. Properties Verification

We describe some security properties as well as their assertion descriptions in PAT code and give the verification result. We do not give a description of the intruder disguised as the data owner, because the disguise as a data owner has no realistic meaning. Because the data owner can be any user and intruders will not use this protocol to get its own data. We use *System* to represent the original model.

Property 1: Deadlock Freedom

The model should not run into a deadlock state. In PAT, there is a primitive to describe this situation:

$$\#assert \text{System deadlock free};$$

Property 2: User Faking

This property represents that the intruder has successfully pretended to be a legal user without being realized by the system. We define a boolean variable *user_fake_success* for verification in PAT.

$$\begin{aligned} \#define \text{User_Fake_Success} \\ \text{user_fake_success} == true; \\ \#assert \text{System reaches User_Fake_Success}; \end{aligned}$$

Property 3: Server Faking

Similarly, this property represents that the intruder has successfully

pretended to be the server without being realized by other entities. We define a boolean variable *server_fake_success* for verification in PAT.

```
#define Server Fake Success
    server_fake_success == true;
#assert System reaches Server_Fake_Success;
```

Property 4: Protocol Completeness

Because *PUOWN* is a public key, if the intruder already acquires the *PUOWN*, then the protocol cannot be fully executed. The server receives three different requests when it executes a complete protocol. We use a constant *n* to record whether the server has accepted three requests. We define a boolean variable *protocol_completeness* for verification in PAT.

```
#define Protocol Completeness
    protocol_completeness == true;
#assert System reaches Protocol_Completeness;
```

The verification results are shown in Fig.2. Deadlock freedom is valid which means that the *System* model does not run into a deadlock state. The *User_Fake_Success* property is valid and PAT provides a trace which leads to a state where this property is satisfied. Intruder can implement an event *req_sc* on channel *FakeD* to be a legal user without being realized. Similarly, the *Server_Fake_Success* property is valid shows that intruder can successfully pretend to be a legal server without being realized. The third property *Protocol_Completeness* is not valid. It represents that intruder can obtain data without executing the complete protocol. Property 2, 3 and 4 verified the insecurity of TESAC.

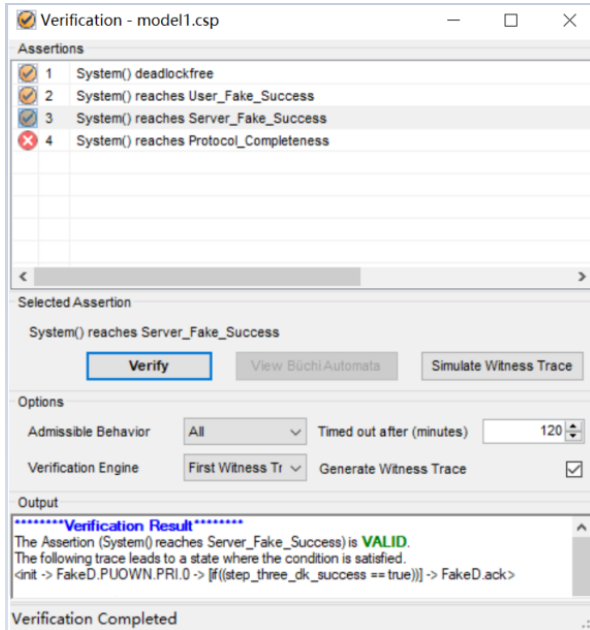


Fig. 2. Verification Result of the model

C. Attack and Improvement

As the verification result shows above, although this scheme adopts asymmetric encryption and the server uses the certificate to authenticate the user, the system is still not reliable. The server

only requests users to obtain the data decryption key and certificate from the data owner for decrypting the data at the time of first data accessing. When users initiate a data access request for the subsequent time, the server directly grants the data. So once the intruder successfully obtains data for the first time, there will be no security in the system. An example trace of the intruder acquires data successfully, which is presented as below:

```
ComUS!msgreq.U.S.req_data ->
FakeU?msgkey1.S.I.E(PUI, E(PRSP, PUOWN)) ->
FakeD!msgkey1.IU.D.E(PUOWN, E(PRI, req_sc)) ->
ComDS?msgreq.D.S.cof -> FakeD!msgreq.IS.D.ack ->
FakeU?msgkey1.D.U.E(PUI, E(PROWN, (s, c)))
```

First of all, the user sends a data request to the server by channel *ComUS*. The intruder intercepts the message sent by the server to the user through channel *FakeU*. Then the intruder pretends to be a user to request the secret and the certificate from the data owner. Data owner validates the users authenticity from the server, this message is intercepted by the intruder and returns a positive feedback message to the data owner. Finally, the data owner provides the certificate and secret key to the intruder.

That is to say, once an intruder obtains a legitimate identity using the above path, the intruder can obtain data before deleting it from the servers user list, which will undoubtedly cause disaster. Next we will make changes to the protocol and not to reduce its time efficiency.

The request information sent by the user for *PUOWN* cannot confirm the origins, as well as the confirm feedback information sent by the server to data owner. In order to change this situation, we will improve the protocol. When the user requests to obtain *PUOWN*, it needs to sign with his own private key, in this way the server can use the digital signature to authenticate the user. Similarly, the confirm message sent by data owner to server and the feedback message of server return to data owner are all signed with their own private key. The specific updating of the model are as follows:

$$MSG_{req} = \{msg_{req}.a.b.E(K_1, E(K_2^{-1}, req_data)), \\ msg_{req}.a.b.E(K_1, E(K_2^{-1}, cof)) | a, b \in Entity, \\ K_1 \in PUKey, K_2^{-1} \in PRKey, req, cof \in Content\}$$

$$MSG_{ack} = \{msg_{ack1}.a.b.E(K_1, E(K_2^{-1}, x)) | \\ a, b \in Entity, x \in Ack, K_1 \in PUKey, K_2^{-1} \in PRKey\}$$

The message changed in the new model corresponds to actions 1, 4 and 5 in Fig. 1. These messages are all related to the server. So we give the updated server process which can be formalized as follow:

```
SERVER0 =df Initialization{n = 0} ->
ComUS?msgreq.U.S.E(PUSP, E(PRUSR, req_data)) ->
GetS!msgkey2.E(PUSP, E(PRUSR, req_data))
.PRSP.PUUSR -> GetS?msgackin.req_data ->
ComUS!msgkey1.S.U.E(PUUSR, E(PRSP, PUOWN)){n = 1} ->
ComDS?msgreq.D.S.E(PUSP, E(PROWN, cof)) ->
GetS!msgkey2.E(PUSP, E(PROWN, cof))
.PRSP.PWOWN -> GetS?msgcont.cof ->
ComDS!msgack.S.D.E(PUOWN, E(PRSP, ack)){n = 2} ->
ComUS?msgkey1.U.S.E(PUSP, E(PRUSR, c)) ->
GetS!msgkey2.E(PUSP, E(PRUSR, c)).PRSP.PUUSR ->
GetS?msgackin.c ->
ComUS!msgdata1.S.U.E(PUUSR, E(PRSP, E(s, data))){n = 3} ->
SERVER0;
```

We define a variable *n* to record the number of messages sent by the server. First, *SERVER₀* initializes the variable *n*. It will be assigned a value of 1 when the server sends *PUOWN* to the

user by channel *ComUS*. *SERVER_0* communicates with its internal process *PROCESSSS* through channel *GetS*. The server will return a feedback message to the confirmation message sent by the data owner. If this action completes successfully, the value of *n* becomes 2. The server provides the data to the user if the user's presented certificate is matched with the corresponding certificate of the requested data. Meanwhile, the value of *n* is modified to 3.

Then we update *ProcessS* to be *PROCSSS* correspondingly.

$$\begin{aligned}
& PROCESSSS =_{df} \\
& \text{GetS?msg}_{key2}.E(PUSP, E(PRUSR, req_data)). \\
& PRSP.PUUSR \rightarrow \\
& \left(\begin{aligned} & \text{GetS!msg}_{ackin}.req_data \rightarrow PROCESSSS \\ & \triangleleft ((PUSP == PRSP) \wedge (PUUSR == PRUSR)) \triangleright \\ & (\text{GetU!msg}_{ackin}.NO \rightarrow PROCESSSS) \end{aligned} \right) \\
& \square \text{GetS!msg}_{key2}.E(PUSP, E(PROWN, cof)).PRSP.PUOWN \rightarrow \\
& \left(\begin{aligned} & \text{GetS!msg}_{ackin}.cof.YES \rightarrow PROCESSSS \\ & \triangleleft ((PUSP == PRSP) \wedge (PUOWN == PROWN) \wedge \\ & (n == 1)) \triangleright (\text{GetS!msg}_{ackin}.cof.NO \rightarrow \\ & PROCESSSS) \end{aligned} \right) \\
& \square \text{GetS?msg}_{key2}.E(PUSP, E(PRUSR, c)).PRSP.PUUSR \rightarrow \\
& \left(\begin{aligned} & \text{GetS!msg}_{ackin}.c \rightarrow PROCESSSS \\ & \triangleleft ((PUSP == PRSP) \wedge (PUUSR == PRUSR)) \triangleright \\ & (\text{GetS!msg}_{ackin}.cof.NO \rightarrow \\ & PROCESSSS) \end{aligned} \right)
\end{aligned}$$

PROCSSS receives the message sent by *SERVER_0* through the channel *GetS*. It should be noted that if the message is a confirmation message, *PROCSSS* will judge whether the value of *n* is 1. This is to confirm that the user is getting PUOWN from the server instead of others.

D. Overall Modeling

We formalize the new system as below.

$$\begin{aligned}
& SYSTEM = SYSTEM_1[INTRUDERPATH]INTRUDER, \\
& SYSTEM_1 =_{df} USER[DATAOWNER]SERVER, \\
& USER =_{df} USER_1[PROCESSPATH]PROCESSU, \\
& DATAOWNER =_{df} \\
& \quad DATAOWNER_1[PROCESSPATH]PROCESSD, \\
& SERVER =_{df} \\
& \quad SERVER_1[PROCESSPATH]PROCESSS.
\end{aligned}$$

The verification results are given as below:

The verification results show that **User_Fake_Success** and **Server_Fake_Success** properties are invalid, which means that system realized that the intruder were performing actions of invading and immediately stopped the process. The **Protocol_Completeness** property is valid, which means that after we improve the protocol. It can be guaranteed that the protocol is executed completely once when the user requests data.

E. CONCLUSION AND FUTURE WORK

In this paper, we have formalized the TESAC using classical process algebra language CSP. Then, we have fed the model into the model checker PAT and verified the security of the model by asserting four properties (deadlock freedom, user faking, server faking, protocol completeness). The verification results show that user faking property and server faking property are valid. It means intruder can successfully pretend to be a legal user or server without being realized. Protocol completeness property is not valid. It demonstrates that TESAC cannot guarantee users to follow all the steps for data accessing when the intruder attacks. This means that TESAC is not secure in a cloud computing environment. In order to solve these problems, we have made improvements to TESAC by introducing a method similar to digital signature. We have verified the improved

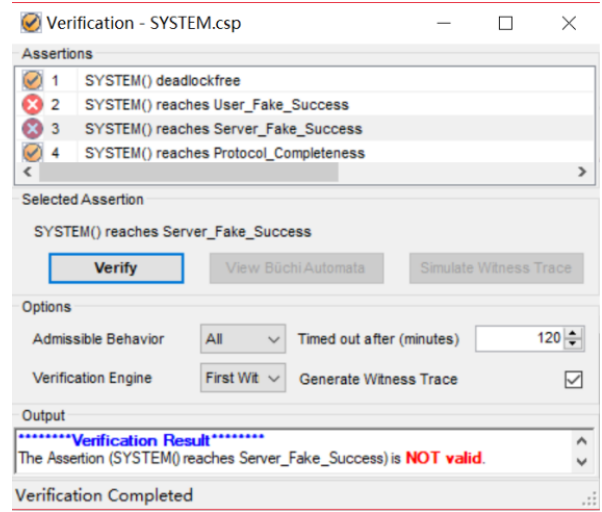


Fig. 3. Verification Result of the improved model

model with respect to the four properties, and the new verification results indicate that the improved model can prevent intruders from invading the system. The security and robustness of TESAC would be improved through our efforts.

ACKNOWLEDGMENT

This work was partly supported by National Natural Science Foundation of China (Grant No. 61872145) and Shanghai Collaborative Innovation Center of Trustworthy Software for Internet of Things (No.ZF1213).

REFERENCES

- [1] M. ARMBRUST, "Above the clouds : A berkeley view of cloud computing," *Science*, vol. 53, pp. 07–013, 2009.
- [2] D. F. Ferraiolo and D. R. Kuhn, "Role-based access controls," *CoRR*, vol. abs/0903.2171, 2009.
- [3] X. W. Gao, Z. M. Jiang, and R. Jiang, "A novel data access scheme in cloud computing," vol. 756. Trans Tech Publications, 10 2013, pp. 2649–2654.
- [4] D. Chen, X. Huang, and X. Ren, "Access control of cloud service based on UCON," in *Cloud Computing, First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings*, 2009, pp. 559–564.
- [5] S. Namasudra and P. Roy, "Time saving protocol for data accessing in cloud computing," *IET Communications*, vol. 11, no. 10, pp. 1558–1565, 2017.
- [6] Z. Mahmood, "Continued rise of the cloud," 2014.
- [7] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [8] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe, "A theory of communicating sequential processes," *J. ACM*, vol. 31, no. 3, pp. 560–599, 1984.
- [9] PAT, "Pat: Process analysis toolkit." [Online]. Available: <https://doi.org/10.1145/828.833>
- [10] A. W. Roscoe, *Understanding Concurrent Systems*, ser. Texts in Computer Science. Springer, 2010.
- [11] G. Lowe and A. W. Roscoe, "Using CSP to detect errors in the TMN protocol," *IEEE Trans. Software Eng.*, vol. 23, no. 10, pp. 659–669, 1997.
- [12] Y. Fei and H. Zhu, "Modeling and verifying NDN access control using CSP," 2018, pp. 143–159.

PRISM Code Generation for Verification of Mediator Models

Weidi Sun and Meng Sun

LMAM & DI, School of Mathematical Sciences, Peking University, Beijing, China

{weidisun, sunm}@pku.edu.cn

Abstract—Component-Based Software Engineering (CBSE) has played an important role in software industry for several decades. The Mediator language is proposed to formally model complex hierarchical component-based systems, which provides a proper automata-based formalism for specifying both high-level system layouts and low-level behavior units. In this paper, we develop a framework for translating Mediator models into the model checker PRISM, and build such a “translator” which can generate PRISM codes from Mediator models automatically and cooperates with PRISM to verify properties of Mediator models. **Keywords:** Mediator, PRISM, Code generation, Model checking, Verification

I. INTRODUCTION

Component-based software engineering has played an important role in software industry for several decades. Implementation details inside components are encapsulated and different components are composed together to construct value-added systems through interfaces. Mediator [10] is a hierarchical modeling language that provides proper formalism for both high-level system layouts and low-level automata-based behavior units of component-based software systems, together with a full-featured type system and powerful coordination mechanisms. However, having powerful modeling languages does not mean correctness of the system models. In practice, errors can still be introduced by the modeling activities. Therefore, we need to investigate the formal analysis and verification techniques for Mediator to guarantee the correctness and reliability of Mediator models.

Model checking [5] is a widely adopted technique for verification of both hardware and software systems. PRISM [9] is a probabilistic model checker that provides a specification language based on the Reactive Modules formalism [1] and a powerful tool support for model checking properties specified in different temporal logics such as LTL, CSL, PCTL, PCTL*, etc. [14]. It has been successfully applied in many areas, including network protocols [7], security protocols [8], coordination languages [4], and so on.

In this paper, we present a framework for translating Mediator models into PRISM such that we can make analysis and verification of Mediator model behavior by using the probabilistic model checker PRISM. This is a further extension to our previous work on the Mediator language [10] and its Arduino C code generation [11].

The following of this paper is organized as follows: After this general introduction, we briefly review some main concepts of the Mediator modeling language in Section II. In Section III, we show how different elements in Mediator can be translated into PRISM. Finally, Section IV concludes the paper and discusses some future work.

II. A MEDIATOR PRIMER

In this section, we briefly review the primary concepts in the Mediator language which concentrates on both high-level system layouts and low-level automata-based behavior units. More details about the language and its semantics can be found in [10], [12].

The syntax tree of a Mediator program is defined as follows:

$$program ::= (typedef \mid function \mid automaton \mid system)^*$$

Typedefs. Mediator provides various data types that are widely used in different formal modeling languages and programming languages. Basic types such as *Integer*, *Bounded Integer*, *Boolean* and *Enumeration* can be easily used to define new data types in Mediator.

Functions. Functions can be either *common functions* which have both interfaces describing inputs and return types of the functions, and function bodies specifying the behavior of functions, or *native functions* that have no function bodies but only interfaces. More discussions about functions can be found in [10].

Automata. *Automata* and *system* are the core modeling elements in Mediator. They are also called *entities* or *components* in a Mediator program. The syntax of *automata* is shown as follows.

<i>automaton</i>	::=	automaton <i>template</i> [?] <i>identifier</i> (<i>port</i> [*]) { (variables { <i>varDecl</i> [*] } [?]) transitions { <i>transition</i> [*] } }
<i>port</i>	::=	<i>identifier</i> : (in out) <i>type</i>
<i>transition</i>	::=	<i>guardedStmt</i> group { <i>guardedStmt</i> [*] }
<i>guardedStmt</i>	::=	<i>term</i> \rightarrow (<i>stmt</i> { <i>stmt</i> [*] })
<i>stmt</i>	::=	<i>assignStmt</i> <i>iteStmt</i> sync <i>identifier</i> ⁺
<i>assignStmt</i>	::=	<i>term</i> := <i>term</i>
<i>iteStmt</i>	::=	if (<i>term</i>) <i>stmt</i> ⁺ (else <i>stmt</i> ⁺) [?]
<i>varDecl</i>	::=	<i>identifier</i> : <i>type</i> (init <i>term</i>) [?]

An *automaton* consists of four parts: *templates*, *interfaces*, *local variables* and *transitions*, which are interpreted as follows:

- 1) *Templates*. Templates include a set of parameter declarations. A parameter can be either a type or a value.
- 2) *Interfaces*. Interfaces consist of directed ports and describe how automata interact with their contexts. Ports can be regarded as structures with three fields: *value*, *reqRead*, and *reqWrite*, which correspondingly denote the values of ports, the status of reading requests and the status of writing requests.
- 3) *Local Variables*. Each automaton contains a set of local variables.
- 4) *Transitions*. The behavior of an automaton is defined by guarded transitions. Each transition consists of a boolean term guard and a sequence of statements. Transitions encapsulated in a group are not ruled by priority in Mediator. In other words, when the guards of two transitions are both satisfied we cannot decide which transition occurs. However the *stmts* in a *guardedStmt* are ruled by priority, they will occur according to the order in the sequence of statements.

The following three types of statements are supported by our translation framework:

- 1) Assignment statement, including an expression and an assignment target, that evaluates the expression and assigns the result to its target if possible,
- 2) Ite (if-then-else) statement that acts as a conditional choice statement in other programming languages,
- 3) Synchronizing statement, labeled with *sync*, that are the flags requiring synchronized communication with other entities.

Systems. A *system* organizes its sub-entities which can be *automata* or *systems*. The syntax of *system* is as follows:

<i>system</i>	::=	system <i>template</i> [?] <i>identifier</i> (<i>port</i> [*]) { (internals <i>identifier</i> ⁺)? (components { <i>componentDecl</i> [*] } [?]) connections { <i>connectionDecl</i> [*] } [?] }
<i>componentDecl</i>	::=	<i>identifier</i> ⁺ : <i>systemType</i>
<i>connectionDecl</i>	::=	<i>systemType</i> <i>params</i> (<i>portName</i> ⁺)

Besides the *templates* and the *interface*, a *system* contains the following parts:

- 1) *Components*. Entities can be placed and instantiated in systems as components. Each component is considered as a unique instance and executed in parallel with other components and connections.
- 2) *Connections*. Connections are used to connect the ports of the system itself, the ports of components and the internal nodes. Inspired by the coordination language Reo [6], [3], [2], complex connection behavior can also be determined by other entities.
- 3) *Internals*. Sometimes we need to combine multiple connections to perform more complex coordination behavior. Internal nodes declared in **internals** segments are untyped identifiers which are capable to weld two ports with consistent data-flow direction.

III. FROM MEDIATOR TO PRISM

In this section we introduce our framework for translation from Mediator to PRISM. The aim of this work is to make analysis of Mediator model behavior by using the probabilistic model checker PRISM.

A Mediator entity will be translated into a module in PRISM. First of all, we consider the “Flat” algorithm which was proposed in [10] and can be used to flatten a hierarchical system into a canonical automaton. The syntax of canonical automata is as follows:

<i>automaton</i>	::=	automaton <i>identifier</i> () { (variables { <i>varDecl</i> [*] }) transitions { <i>transition</i> } [*] }
<i>transition</i>	::=	group { <i>guardedStmt</i> [*] }
<i>guardedStmt</i>	::=	<i>term</i> \rightarrow (<i>stmt</i> { <i>stmt</i> [*] })
<i>stmt</i>	::=	<i>assignStmt</i> <i>iteStmt</i>
<i>assignStmt</i>	::=	<i>term</i> := <i>term</i>
<i>iteStmt</i>	::=	if (<i>term</i>) <i>stmt</i> ⁺ (else <i>stmt</i> ⁺) [?]
<i>varDecl</i>	::=	<i>identifier</i> : <i>type</i> (init <i>term</i>) [?]

Such a flattening of Mediator system model has been proven to be valid, and the syntax of the resulting canonical automaton is similar to the corresponding PRISM model defined by a module as follows:

```

model ::= module identifier
        declaration+;
        (transition;) *
endmodule

```

Both the variables declarations and guarded statements for the transitions in an automaton can be easily mapped to the corresponding PRISM model as well.

In our framework, we have six components that work together to generate PRISM code from Mediator models: *entity generator*, *typedef generator*, *term generator*, *virtual term generator*, *transition generator* and *automaton generator*. We will show details about these generators, especially the last two, in this section.

A. Generators for Entity, Typedef, Term and Virtual Term

The *entity generator* is designed for calling the algorithm for flattening and fed the returning canonical automaton to the *automaton generator*. The canonical automaton has no parameters and ports which indicates that it does not communicate with the environment.

With the help of *typedef*, we can give an alias to an existing definition to simplify the expression in Mediator. Although we cannot use the *typedef* and alias in PRISM, the absence of similar syntax can be overcome by using the original definitions directly. The *typedef generator* returns a map which maps aliases to original definitions and helps us find the original definition when meeting an alias.

There are some slight differences between terms in Mediator and PRISM. For example, in Mediator we use “==” to denote the equality operator, while in PRISM “=” is used instead.

Term generator and *virtual term generator* are designed for dealing with such subtle distinctions and the only difference between them is that the latter, as the name suggested, is for virtual terms. More details of these two generators can be found in [13].

B. Transition Generator

The transition generator is designed for generating transitions. There are two main differences between Mediator transitions and PRISM transitions:

- In Mediator, the *stmts* in a *guardedStmt* are ruled by priority. However, in PRISM we do not have priority because of its assignment method. For example, if we have $a = 0, b = 0$ and then make the assignment a, b : ' $a = 1, b = a$ '. In Mediator the result is $a=1, b=1$, because we first assign the value 1 to a and then assign the value of a , which has been changed to 1, to b . In PRISM the result is $a=1, b=0$ which is completely different. Because the assignment statement is ' $(a = 1) \& (b = a)$ '. We assign a', b' at the same time and then assign values of a', b' to a, b .
- The *ite* statements can be nested in Mediator, but this is not permitted in PRISM.

Transition generator cannot work while ignoring these two problems. To solve the first problem we take the transition apart and execute the transitions sequentially. Though the execution of transitions in PRISM does not have priority, we can create the priority by adding a "pedometer" to guards. A new variable *tranmark* is provided, which is a counter to record the number of executed *stmts*. For every transition with at least two *stmts*, we separate it into several transitions. In each new transition, the new *stmt* contains one original transition's *stmt* and an assignment statement: " $\text{tranmark_i} = \text{tranmark_i} + 1$ ", the new guard consists of the original transition's guard and a condition: " $\text{tranmark_i} = n$ ". Fig. 1 shows such an example of transition separation.

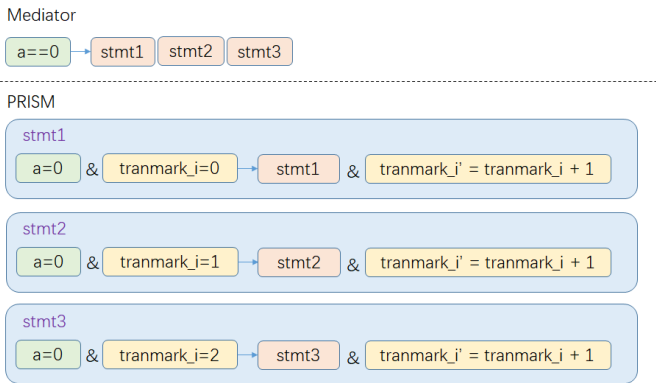


Fig. 1. Separation of transitions

In Fig. 1, we give every new transition a *tranmark_i* and initialize it to 0. When we want to execute a transition, the first new transition's guard " $a = 0 \& \text{tranmark_i} = 0$ " must be satisfied. If it is satisfied we can execute *stmt1* and add 1

to *tranmark_i*. After that, the second new transition's guard will be satisfied, and *stmt2* will be executed. These steps will be repeated until " $\text{tranmark_i} = n$ ". Following these steps we can execute every statement once and only once in order and the priority will be guaranteed.

However only separating the transitions is not enough for our framework. When we finish executing *stmt1* (for example *stmt1* is " $a' = a + 1$ "), the value of variable a may change, and the next new transition's guard " $a = 0$ " may not be satisfied. Here we need a new concept *virtual variable* to replace " a " in calculation. For example, " $a' = a + 1$ " in the transitions will be substituted with " $v_a' = v_a + 1$ " so that the original variables in guards will not change. Once all the executions of the original transition are finished, we assign the value of the *virtual variables* to the original variables. An example of such virtual variables for transition separation is shown in Fig. 2.

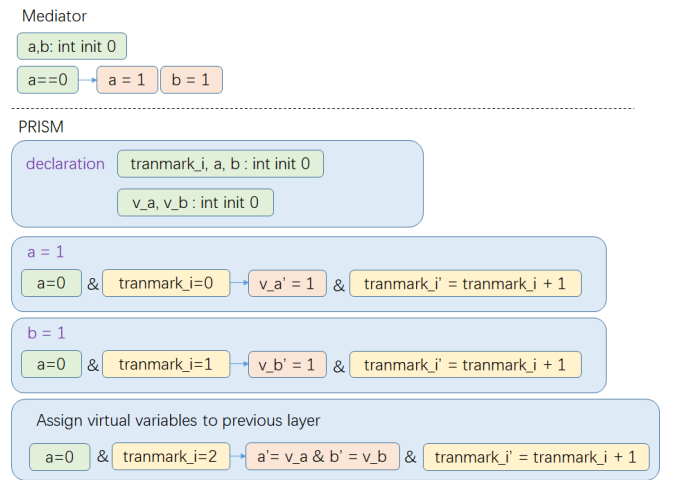
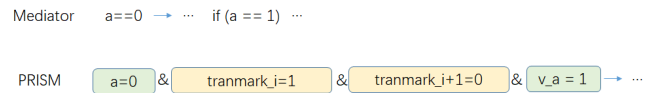


Fig. 2. Virtual variables for transition separation

The nesting problem for *ite* statements is entangled with the priority problem and thus more complicated. In other words, in the recursive generation, the nested *iteStmt* being treated as a new transition (regard the condition of *iteStmt* as a guard) shares all the troubles of the transition, the priority problem is no exception. We introduce a new variable: *layer* to denote the number of the current *iteStmt*'s nesting layers.

The new transition's guard generated from the nested *iteStmt* will be the combination of the old guard and the *iteStmt*'s condition. Furthermore, we will give every *iteStmt* a new *tranmark* as well, for example:



Once a new transition for *iteStmt* is created, the generating process will enter a new layer, and the value of *layer* will be increased by 1. A set of new *virtual variables* corresponding to the *virtual variables* in the previous layer is also needed so that the change of variable values does not affect the

satisfiability of the new guard which contains the *iteStmt*'s condition. The generating process will exit the current layer when the execution of the *iteStmt* is completed, and the value of *layer* will be decrease by 1. We also introduce a variable *maxlayer* to record the largest *layer* that appeared. Combining with the above solution the example is shown in Fig. 3.

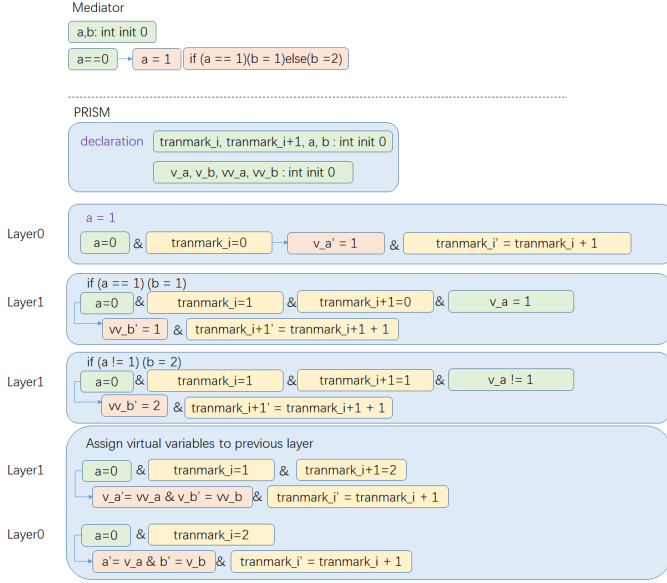


Fig. 3. Example of layer solution

The transition generator is designed as a recursive function *TGF*, which is invoked when we generate a transition and terminates when the generation process finishes. If we meet an *iteStmt* on the sidelines of a *TGF* execution, a new *TGF* will be invoked inside the old one.

To put it in a nutshell, the transition generator treats each statement as an atomic operation, i.e., an operation which cannot be interrupted and executes them in order.

C. Automaton Generator

The *Automaton Generator*'s goals are two-fold: adding the *global declarations* and combining different parts of the generated PRISM model.

Most types in *global declarations* which are supported by the PRISM language are easy to define. For these types, the only work we need to do is changing some grammar formats in Mediator's definitions. However, it does not work for *EnumType* and *ListType*. The solving of *EnumType* and *ListType* are similar, for *EnumType*, we define *IntType* variables for every identifier in it and initialize them to 0,1,2... in order. For *ListType*, we define *IntType* variables which is named as "ListName*i*" for every element in the list. It needs to be pointed out that the list we defined is a fixed-length list and all the elements in it are initialized to 0. Besides, before defining such variables we need to change the user-defined types to base types, and the changing approach is mentioned in the *typedefGenerator*.

Then we have the model type, the model name, the global declarations and the transitions which can be generated by *transition Generator*; the final step is to combine these parts to build the PRISM model. After finishing all these works, the *automaton Generator* returns the result module in PRISM.

IV. CONCLUSION AND FUTURE WORK

In this paper, we presented a code generator that converts Mediator models to PRISM models. Mediator provides a component-based modeling language in which components and systems can be defined in a hierarchical way. With the help of this code generator, the Mediator models for complex systems can be transformed into PRISM automatically such that properties of the Mediator models can be verified by using the PRISM model checker.

In the future we plan to extend the Mediator language and investigate more quantitative aspects of system models such as reliability, security, etc., in Mediator. Providing support for more hardware platforms and programming languages is in our scope for future work as well.

ACKNOWLEDGEMENTS

The work is partially supported by NSFC under grant no. 61772038, 61532019, 61202069 and 61272160, and the Guangdong Science and Technology Department (Grant no. 2018B010107004).

REFERENCES

- [1] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
- [2] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical structures in computer science*, 14:329–366, 2004.
- [3] Farhad Arbab, Christel Baier, Frank de Boer, and Jan Rutten. Models and temporal logical specifications for timed component connectors. *Software & Systems Modeling*, 6:59–82, 2007.
- [4] Farhad Arbab, Sun Meng, Young-Joo Moon, Marta Z. Kwiatkowska, and Hongyang Qu. Reo2mc: a tool chain for performance analysis of coordination models. In *Proceedings of ESEC/FSE'09*, pages 287–288. ACM, 2009.
- [5] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.
- [6] Christel Baier, Marjan Sirjani, Farhad Arbab, and Jan Rutten. Modeling component connectors in reo by constraint automata. *Science of computer programming*, 61:75–113, 2006.
- [7] Marie Dufflot, Laurent Fribourg, Thomas Héroult, Richard Lassaigne, Frédéric Magniette, Stéphane Messika, Sylvain Peyronnet, and Claudine Picaronny. Probabilistic model checking of the CSMA/CD protocol using PRISM and APMC. *ENTCS*, 128(6):195–214, 2005.
- [8] Salekul Islam and Mohammad Abu Zaid. Probabilistic analysis and verification of the ASW protocol using PRISM. *International Journal of Network Security*, 7(3):388–396, 2008.
- [9] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Proceedings of CAV 2011*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
- [10] Yi Li and Meng Sun. Component-based modeling in mediator. In *Proceedings of FACS 2017*, volume 10487 of *LNCS*, pages 1–19. Springer, 2017.
- [11] Yi Li and Meng Sun. Generating Arduino C Codes from Mediator. In *It's All About Coordination*, volume 10865 of *LNCS*, pages 174–188. Springer, 2018.
- [12] Mediator github repository. <https://github.com/mediator-team>.
- [13] Mediator to PRISM translator. <https://github.com/Weidi-Sun/mediator-master>.
- [14] PRISM. <http://www.prismmodelchecker.org>.

An evolutionary model for dynamic and adaptative service composition in distributed environment

Jiawei Lu, Huan Zhou, Jun Xu, Gang Xiao
School of Computer Science and Technology
Zhejiang University of Technology, Hangzhou, China
Email: {viivan, zhouhuan, xujun, xg}@zjut.edu.cn

Haibo Pan
Whzhen avenue Technology Co., Ltd
Hangzhou, China
Email: 345383630@qq.com

Abstract—Service composition is an important mean for integrating the individual Web services to create new value-added systems that satisfy complex requirements. Therefore, how to effectively analyze different types of services and find out the matching similarity between services to efficiently substitute failed services in a distributed and dynamic environment becomes crucial to service composition. In this paper, we propose a novel approach based on a data cell evolution model (DCEM) to support the dynamic adaptation of service compositions. The model combines data service information and biological cell behavior analysis to encapsulate data services into data cells. In order to reach optimum adaptations, we analyzed the static and dynamic structure of data cells based on bigraph theory to guarantee the consistency of service evolution. To evaluate the proposed approach, a series of simulation experiments and comparisons are conducted to demonstrate the effectiveness of service composition.

Keywords- Service Composition; Data Service; Bigraph; Data Cell; Service Evolution.

I. INTRODUCTION

Data as a Service (DaaS) is a new cloud computing service model that provides consumers on demand with data through different protocols on the Internet in a timely and low cost manner. However, as the function of a single Web service is simple and limited, it is difficult to meet the various requirements in a complex network environment. How to effectively model evolution of service composition and analyze its service behavior has become an issue that existing research must deal with.

Due to Web service with multi-source, heterogeneous, autonomous and dynamic characteristics, the evolution of service composition is different from traditional software evolution and faces more serious challenges. Many scholars have researched in this field through formal methods [1], semantic [2] or combinatorial models [3]. However, the above studies mainly focus on abstract behaviors and semantic analysis of services in specific field, without considering the dynamic contexts. Consequently, they easily lead to performance degradation and composition failure when the environment changes.

Biological cell is a precise structural, functional, and evolutionary unit, which structure can change dynamically with the environment during growth, differentiation and

physiological process. Comparing the dynamic behavior of service composition with biological cell, they have similarities in some aspects. It is possible to combine data service with biological cell to analyze the evolution of service composition [4,5].

In this paper, we encapsulate data services into data cells and analyze the dynamic behavior at the cell level. The data cell, like biological cell has a strong hierarchical structure. Thus, a formal method is needed to effectively explain the static and dynamic information of data cells and reflect the important characteristics such as functions and location interconnectivity of services. Bigraph [6] is a graphical formalization theory tool. It has more extensive applications in formal modeling and consistent evolutionary analysis [7].

In order to reason about the evolution of service composition better, we propose a data cell evolution model (DCEM) to increase the flexibility of service in Web system. The main contributions of this paper can be summarized as follows:

- We use bigraph theory to encapsulate data services into data cells, and construct evolution model for data cell to describe different information of service such as service name, service quality, service context, and so on.
- Because the composition processes may be canceled, or services may be moved or withdrawn, it is necessary to recombine it to provide a more powerful service, so that the service dynamic behavior based on bigraphical reactive system is analyzed. Meanwhile, we propose a self-healing algorithm which is used to dynamically adjust available service to substitute the failed ones.

II. DATA CELL MODELING BASED ON BIGRAPH THEORY

A. The Bigraph Theory

Bigraph is proposed by Milner and other scholars in 2001 to emphasize the location and connection of computing (physical or virtual) [6]. Bigraph is a 2-tuple $B = \langle B^P, B^L \rangle$. B^P is the place graph and B^L is the link graph. The place graph is used to represent the location of nodes, which are nested with each other in bigraph. The link graph ignores the nested relationship and only indicates the connection between nodes. Fig. 1 shows a bigraphical structure.

DOI reference number: 10.18293/SEKE2019-120

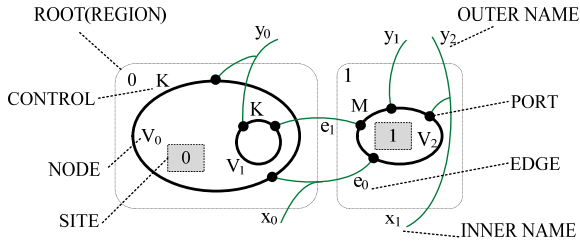


Fig. 1. Elements of bigraph

B. Data Cell Modeling

Based on the bigraph theory, we constructed DCEM that maps the structures and message interaction of services to bigraphs, so as to formalize the services and their compositions by process calculus. DCEM mainly consists of two layers: data cell and data cell cluster. The form is defined as follows:

Definition 1 (Data Cell). A bigraph definition of data cell is a 5-tuple $DC = \langle S, E, Ctrl, C^p, C^l \rangle: \langle m, X \rangle \rightarrow \langle n, Y \rangle$, where:

- S is a limited set of services in a data cell, $\forall s \in S$ is called a data service;
- E is a set of finite edges, $\forall e \in E$ is called a connecting edge;
- $Ctrl: S \rightarrow C$ is a mapping relation between services and service controls;
- C^p is the place graph to represent the location of services and C^l is the link graph to represents service dependencies;
- The inner interface $\langle m, X \rangle$ indicates that the bigraph has m sites and a set of inner names X . The outer interface $\langle n, Y \rangle$ indicates that the bigraph has n regions and a set of outer names Y .

Definition 2 (Service Control). A service control is a 5-tuple $C = \langle CN, CT, P, CL, U \rangle$, where:

- CN is a control name of service. CT is used to specify the type of this service, whether atomic or composite;
- P is a limited set of ports, which describes the inputs and outputs of service, $\forall p \in P$ is called a service port;
- $CL = \langle DL, CN \rangle$ is the dependency status of current service, including DL which is the dependent level with CN from other service;
- U is a probability value which represent the service reliability.

Definition 3 (Bigraphical Reactive System). A bigraphical reactive system for data cell is a 3-tuple $BS = \langle BC, R, BC' \rangle: BC \rightarrow BC'$, where:

- BC is the reactants and BC' is the products, which are corresponding to data cells with the bigraphical structures;
- R is a set of reaction rules and specifies the reaction process from BC to BC' .

As examples depicted in Fig. 2, the bigraph definition of data cell aims to represent the structural relationship and data characteristics in services. The core element correspondence between data cell and bigraphical structure is also shown in Table I.

TABLE I. DATA CELL STRUCTURE DEFINITION

Element in data cell	Element in bigraph	The example
DC	Root	$0, 1, \dots$
m	Site	s_1, s_2, s_3, \dots
S	Node	s_1, s_2, s_3, \dots
CL	Edge	e_0, e_1, e_2
C	Node control	C_1, C_2, C_3, \dots
pC	Node Ports	$\bullet, \bullet, \bullet$

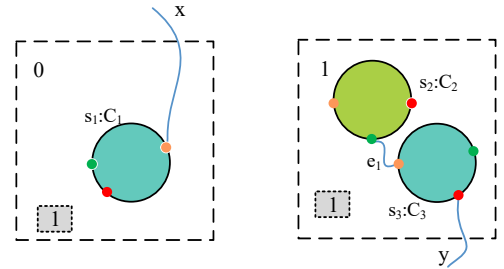


Fig. 2. Bigraph form of data cells

In practice, in order to meet the increasingly complicated requirements of users, it is necessary to select appropriate services from the network and combine them according to certain business rules to construct a scalable, loosely coupled combination. The data cell cluster is based on four kinds of structures (sequence, conditional, parallel, and loop) in service composition and combines a plurality of data cells with bigraph operations. The relevant forms of data cell cluster proposed in this paper are as follows:

Definition 4 (Data Cell Cluster). A bigraph definition of data cell cluster is a 3-tuple $DCC = \langle DCS, CS, LinkS \rangle$, where:

- DCS is a limited set of data cells;
- CS is a limited set of composite structures in data cells;
- $LinkS$ is a limited set of link ports in the cluster, $\forall Link \in LinkS$ is called a connection between two ports.

The term language [13] is the basis for the formal specification and verification of dynamic evolution in bigraph. Fig. 3 shows the structure of DCC based on different workflows. Taking the parallel structure (Case 3) as an example, the cluster has three data cells DC_0, DC_1 , and DC_2 . The place graph indicates the positional relationship of services and other information (e.g., the number and distributivity of the cells). The link graph shows the dependency relationship of the services.

To adapt the dynamic environments to complex requirements, the data interaction between cells is constantly changing, forming new cell clusters or modifying the original cell cluster structure. Ensuring the structural integrity of data cells during this interactive process, while increasing the effectiveness of service composition, requires serious consideration. A checking technique for verifying the data flow of the process model and re-adjusting the model

according to the feedback has been proposed [14]. However, this process model has a large detection granularity and may easily detect distortion. We present the bigraph matching algorithm (Algorithm 1) to evolve data cells according to a bigraphical reactive system (see definition 3). During matching, the constraints of R in the bigraph are dynamically determined by the context and requirements. A reaction rule in R specifies the reaction process, and can take any number of parameters. Finally, a new bigraph is generated when the data cells match successfully.

The algorithm contains two phases and takes into account time, QoS, and service context information constraints. In the first phase, we take an initial bigraph BC and a set of reaction rules R . For each reaction rule r in R , the method $isMatch(BC, r)$ is called to determine whether the elements in the bigraph can be matched. In the second phase, if r is matched and the constraint is satisfied, the matching part in the bigraph will be replaced by products in r . $isMatch(BC, r)$ is a recursive method that is iteratively executed until the last node in the bigraph has been checked.

Algorithm1 BigraphMatch

Input: bigraph BC (an initial bigraph), a set of reaction rules R

Output: a new bigraph BC'

```

1: if  $R == \text{Null}$  then
2:   return  $BC$ 
3: else
4:   for each reaction rule  $r$  in  $R$  do
5:     flag = isMatch( $BC, r$ )
6:     if (flag == TRUE & & timeConstraints == true) then
7:       // If the match is successful and satisfies the time constraint, the
       // reaction proceeded
8:        $BC' = BC \cup \{BC \mid \text{the matching part in } BC \text{ with reaction in } r\}$ 
9:       return  $BC'$ 
10:    end if
11:  end for
12: Procedure isMatch( $BC, r$ )
13: Input: bigraph  $BC$ , term  $r$  in  $R$ 
14: Output: a flag to indicate whether the match is found
15: 1: for each service  $s$  in  $BC$  do
16:   2: if ( $r$  contains  $s$ ) then
17:   3: if ( $s.CN \neq r.CN \parallel s.CT \neq r.CT \parallel s.CP \neq r.CP$ ) then
18:     // Service control matching
19:     continue
20:   4: else
21:     for each port  $p$  in  $C$  do // Service port matching
22:       5: if ( $s.pI \neq r.pI \parallel s.pN \neq r.pN \parallel s.pT \neq r.pT \parallel$ 
23:          $s.pC \neq r.pC$ ) then
24:         6: continue
25:       7: else
26:         return True
27:       8: end if
28:     end if
29:   9: end if
30: 10: else
31:   continue
32: 11: end if
33: 12: end for
34: 13: return False

```

III. EVOLUTIONARY ANALYSIS OF SERVICE DYNAMIC BEHAVIOR

In actual use, service composition may face situations such as service failure and service composition disorder. These cause data cell variation, thus losing the original functional properties and structural stability.

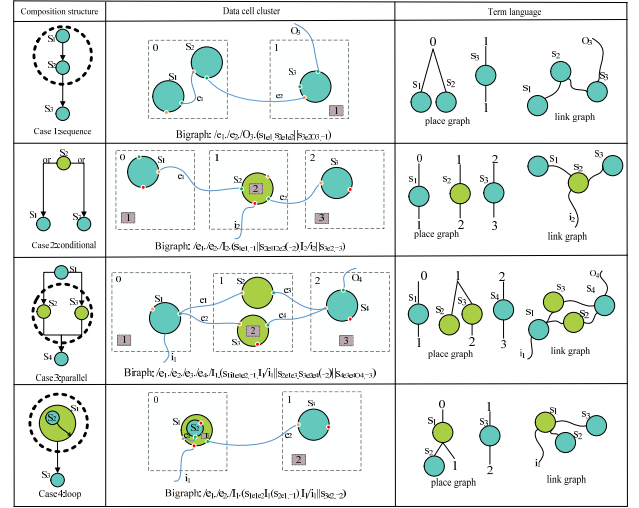


Fig. 3. The basic structures of data cell cluster

In our work, DCEM periodically tests the service availability through a data cell self-healing algorithm (Algorithm 2), allowing the structural variation of the cells to be fixed. This enables the service composition to restore the expected functions, and ultimately achieves the effect of self-repair to improve the service adaptability. The basic process of Algorithm 2 is as follows:

Algorithm2 Data Cell Self-Healing

Input: bigraph BC (an initial bigraph)

Output: a new bigraph BC'

```

1: for each service  $s$  in  $BC$  do
2:   flag = isInvalid( $s$ ) // indicate whether the services is invalid
3:   if (flag == true) then
4:     a broken bigraph  $BC^* = \text{BigraphReplace}(BC, s)$ 
5:     // Choose the most reliable service  $MDCC$ 
6:      $MDCC = \text{Max}(\text{CR}(DCC))$ 
7:     generate new reaction rules  $R = \text{createR}(s, MDCC)$ 
8:     // use algorithm 1 to ensure the structural integrity of DC
9:      $BC' = \text{BigraphMatch}(BC^*, R)$ 
10:   end if
11: end for

```

- Call method $isInvalid(s)$ at a specified time interval to check whether the service in data cell is fail.
- If the service fails then use Algorithm 3 to adjust the bigraphical structure, such as delete nodes and control belong to the failed service. Otherwise adjust the time interval to continue testing.
- Select the most reliable service from the similar service clustering to replace the fail one. There are many clustering algorithms; we mainly use the tag clustering algorithm in reference [15] and execute the aggregation process to construct the similar service clustering. Then the chosen service is considered as a reaction in rules R .
- Based on the previous steps and the bigraphical reactive system, generate the appropriate reaction rules R , and call Algorithm 1 to verify the rationality of reaction. Finally, a new bigraph is generated with a new reliable composition.

When service is detected as failed, Algorithm 3 needs to find out the failed service in the data cell. For given bigraphical information and a failed service s , we traverse each node in bigraph. If there is a surjection relationship

between the node and s , the corresponding structure is deleted and a broken bigraph is given. The specific algorithm is described as follows:

Algorithm3 BigraphReplace

Input: bigraph BC (an initial bigraph), s (a failed service)

Output: a broken bigraph BC^*

```

1: for each service  $s'$  in  $BC$  do
  //Indicate whether the service is match
2:   flag = node_conMatch( $s'$ ,  $s$ )
3:   if (flag == true) then
    //Delete structures belong to  $s'$ 
4:      $BC^* = \text{deleteBigraph}(BC, s')$ 
5:     if ( $s'.C.CL.DL \neq \text{single}$ ) then
      // Delete dependency belong to  $s'$ 
6:        $BC^* = \text{deleteDependent}(BC^*, s')$ 
7:   end if
8: end if
9: end for

```

IV. CASE STUDIES AND VERIFICATION

To illustrate the effectiveness of DCEM in service composition, we introduce a composite service that supports online book shopping at Orange Country Bookstore (OCB) [16] (depicted in Fig. 4).

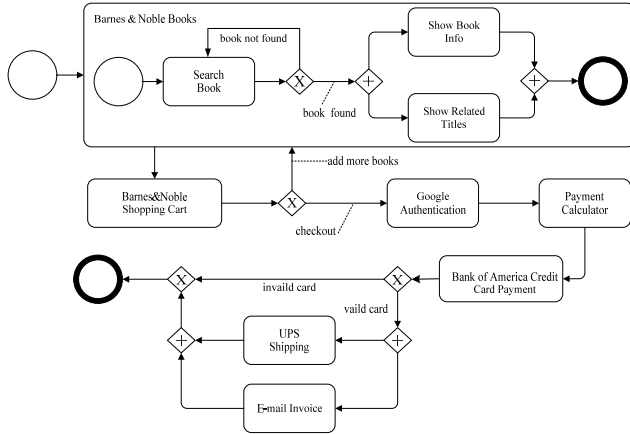


Fig. 4. Composite service for online book shopping

The business process in OCB includes (1) Look for books, including *Search Book*, *Show Book Info*, *Show Related Titles*, which are all part of the *Barnes & Noble Books* composite service. (2) Add books to shopping cart in a loop, e.g., *Barnes & Noble Shopping Cart*. (3) Authentication and payment at checkout, such as *Google Authentication* or *Payment Calculator*. (4) Email and invoice service, such as *invalid Card*, *UPS Shipping Web*, and *E-mail Invoice*.

The service composition created by this instance may change the contextual events during the actual operation, resulting in service failure. In addition, the system requirements state that the service composition must maintain a main workflow after the evolutionary adjustment (for example, always ensuring that books are first searched and then added to the cart).

Here, we fully consider the evolution possibility of each service by analyzing its context. First, we construct the data cells based on the different business processes, as listed in

Table II, and then further evolve the data cells into cell clusters according to functional attributes and requirements. Table III lists the data cell clusters related to the main workflow of the shopping cart, searching for books, and payment in the OCB website.

When *Barnes & Noble Books* is unavailable, causing the functionality of s_4 to go missing (see Fig. 5), the system detects the failed service and traverses the data cell cluster according to Algorithm 3 to alter the structure. It then finds the substitute service *Amazon Books* (DCC_{AB}) from the similar service clustering via Algorithm 2 to repair the structure.

TABLE II. DATA CELL MODELING

Data Service	Data Cells	Term Language	Data Service	Data Cells	Term Language
Search Book		$/m(S_{id}) \rightarrow m(x)$	Show Book Info		S_2
Show Related Titles		S_2	Shopping Cart		$/m(S_{id} \rightarrow m(y))$
Google Authentication		S_5	Payment Calculator		S_6
Bank of America Credit Card Payment		$/m(S_{id}) \rightarrow m(y)$	E-mail Invoice		S_9
UPS shipping		S_9	Book Searching		$/k(S_{id}) \rightarrow k(z)$
Book Description		$/k(S_{id}) \rightarrow k(z)$	Related Titles		S_{12}

TABLE III. DATA CELL CLUSTER MODELING

Data Services	Data Cell Clusters	Term Language
Barnes & Noble Books Shopping Cart		$/e_1/e_2/e_3/e_4/e_5. (s_{1e2e4e5} (s_{1e1e2} s_{2e1e3} s_{3e2e4})) s_{5e5}$
Amazon Books		$/e_1/e_2. (s_{10e1} s_{11e1e2} s_{12e2})$
Credit Card Payment shopping		$/e_1/e_2. (s_{7e1e2} s_{8e1} s_{9e2})$

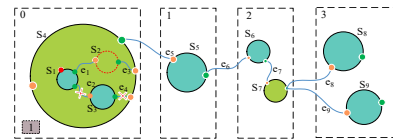


Fig. 5. Data cell cluster on OCB when *Barnes & Noble Books* has failed

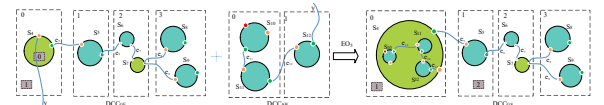


Fig. 6. Self-healing process in data cell cluster with *Amazon Books*

Finally, as shown in Fig. 6, DCC_{AB} is absorbed by the data cell cluster, retaining an unchanged workflow order such as a looping structure with *Book Searching*, *Book Description*, *Related Titles*, and *Shopping Cart*. s_{10} , s_{11} , and s_{12} are added in the site of s_4 . After that, the output port of s_{12} is connected to the input port of s_4 by e_{13} . Tables IV summarize the term language operational sets as EO_I to formalize this process of evolution.

TABLE IV. TERM LANGUAGE OPERATIONAL SET FROM EO_I

DC	EO_I
DCC_{GE}	$EL_{rule}: /n. \rightarrow /e_{13}.$
	$Site: -0 \rightarrow s_{10}e_{11} s_{11}e_{11}e_{12} s_{12}e_{12}e_{13}$
	$x/y: n/y \rightarrow \Phi$
	$\prod_{i=1}^4 DC_{i \text{ ar}(f)}: (s_{4e_5n}(-0) -1 n/y s_{5e_5e_6} s_{6e_6e_7} s_{7e_7e_8e_9} s_{8e_8} s_{9e_9})$ $\rightarrow (s_{4e_{13}e_5}(s_{10e_{11}} s_{11e_{11}e_{12}} s_{12e_{12}e_{13}}) -1 $ $s_{5e_5e_6} -2 s_{6e_6e_7} s_{7e_7e_8e_9} s_{8e_8} s_{9e_9})$
DCC_{AB}	$EL_{rule}: /y. \rightarrow /e_{13}.$
	$U V: s_{10e_{11}} s_{11e_{11}e_{12}} s_{12e_{12}e_{13}} \rightarrow s_{4e_{13}e_5}(s_{10e_{11}} s_{11e_{11}e_{12}} s_{12e_{12}e_{13}})$

These actions express how to reorganize elements in the composition model to re-select suitable data cells from the same cluster when the *Barnes & Noble Books* composite service fails. The failed units are replaced by *Amazon Books* and *Related Titles* (according to the business process from Fig. 4).

The self-healing process in Fig. 6 is formalized by the following term language:

$/e_5./e_6./e_7./e_8./e_9./n.(s_{4e_5n}(-0)|-1|n/y||s_{5e_5e_6}||s_{6e_6e_7}|s_{7e_7e_8e_9}||s_{8e_8}|s_{9e_9})$
 $\rightarrow /e_{11}./e_{12}./e_{13}./e_5./e_6./e_7./e_8./$
 $e_9.(s_{4e_{13}e_5}(s_{10e_{11}}|s_{11e_{11}e_{12}}|s_{12e_{12}e_{13}})|-1||s_{5e_5e_6}|-2||s_{6e_6e_7}|s_{7e_7e_8e_9}||s_{8e_8}|s_{9e_9})$

V. EXPERIMENT ANALYSIS

To illustrate the effectiveness of the algorithm described in this paper, we conducted a series of experiments in different settings. The PC configuration was as follows: Intel Core i5-8250U CPU (1.6 GHz), Windows 8 and 6 GB RAM. We used all 12 of the atomic services discussed in Section 4. We then extracted their parameters and used them as a seed to randomly generate an extended dataset with 500-2000 services. Each service contained basic information such as the service name, input, output, and success probability. The experiments compared three kinds of algorithm: (1) The algorithm (DPSRM) based on the Dynamic Software Product Line approach [17]; (2) The algorithm (IASRM) based on the process ontology and multiple recovery [18]; (3) The data cell self-healing algorithm (DCSRM) proposed in this paper. In addition, we randomly set the effective service that failed as a variant v during the composition, which automatically triggers the service substitution. To ensure unbiased statistical results, all algorithms were executed independently 20 times.

Fig. 7a presents the results obtained using only DCSRM. The service number varies from 100 to 500 and the number of variants increases gradually from 1 to 8. We found that when the number of variants is small, an increase in the service number produces a steady increase in the reliability. However, as the number of variants increases, the reliability becomes relatively low, especially when the number of services is small. This is because there are fewer similar

services in the small service set, resulting in no suitable service being found when the number of variants grows. Fig. 7b shows results for $v = 9$ and the service number varying from 100 to 2000. Initially, IASRM gives the highest reliability, but this obviously decreases as the number of services increases. Furthermore, DCSRM always outperforms DPSRM.

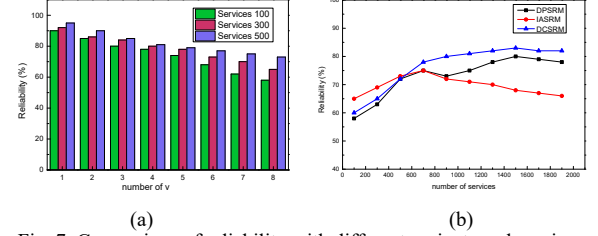


Fig. 7. Comparison of reliability with different variants and services

From Fig. 8, we can see that the response time grows with the number of services and variants. Moreover, no matter how v is allocated, DCSRM performs much better and faster than IASRM and DPSRM. Thus, the experimental results illustrate that our algorithm significantly improves reliability and reduces the time cost of service composition.

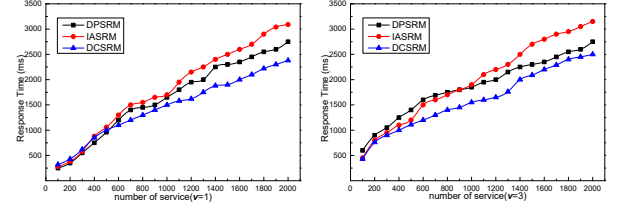


Fig. 8. Comparison of response time with different variants and services

VI. RELATED WORK

A. Bigraphs and their application

Bigraphs were proposed by Milner and other scholars in 2001 to emphasize the location and connection of computing units (physical or virtual) [6], and has now become the tool of choice for many service-adaptive and software-reconfigurable systems because of its complete formal theory and dynamic mobility. For example, Lian et al. [7] simulated modeling using a bigraphical reactive system to analyze the mobile cloud. Calder et al. [8] proposed a model based on checking predicates from user-initiated and network events by extension to a bigraphical reactive system.

B. Evolution of service composition

Ensuring the rationality of business process structures after evolution is an important problem in the composition of services. If the evolution operation is not implemented properly, it may cause problems such as a logical deadlock or component service unreachability [20]. This section analyzes the evolution of service composition in terms of the rationality of service evolution operations and the current solutions for failure recovery.

1) Rationality of service evolution operations

The rationality of the service composition process and the correctness of the data flow are usually guaranteed by defining the evolution criterion in the operation process, thereby avoiding complex verification processes after evolution. To ensure the rationality of the service composition process, Zeng et al. [9] proposed a set of basic evolutionary operations for adjusting the service workflow structure based on workflow network modeling, which guarantees the rationality of the internal process logic during the business process. Urbieto et al. [3] propose an adaptive service composition framework that supports the dynamic reasoning of user requests and service behaviors in the smart city. Khanfir et al. [2] propose a framework for automatic generation and publishing of service descriptions by using OWL-S semantic annotations, the purpose of it is to analyze the process modeling and the choreography of service composite. In service behavior analysis, the research is mainly used by formal methods such as Petri net, process algebra, and π calculus, etc.

2) Current solutions to failure recovery

Self-adaptation is the ability of a system to adapt to changes in its environment to maintain the original functionality, and is used in different problem domains. Many recent studies have focused on enabling adaptation for BPEL processes. For instance, the monitoring mechanism embedded in the BPEL engine can be used to capture fault messages [10], allowing existing processes to be directly deployed without any modifications. Some scholars perform fault monitoring and recovery through transaction attributes of object states. Ettazi et al. [11] fulfilled user requirements in mobile environments by focusing on transactional aspects of context-aware services. In addition, there has been some research based on security monitoring and self-healing systems [12,19]. Asim [12] presented a framework that automates the monitoring of business processes and reports the compliance violations at runtime.

VII. CONCLUSION

Service composition is an important technology for integrating information to create new value in systems that satisfy complex requirements. In this paper we propose a novel approach for service composition with data cell modeling, which is inspired by biological cell and guided by the bigraph theory. This enables users to efficiently analyze the services in a dynamic distributed environment.

However, the preliminary data cells and clusters from model still need to be manually configured by analyzing and extracting the important characteristics from services. So, for future work, some tools may be applied to extract service features automatically. Another problem for future research is that more constraints from specific customer requirements, including service rating, service price and so on, need be considered to optimize the model in different scenes.

ACKNOWLEDGMENT

This work is supported by the Science and Technology Key Research Planning Project of Zhejiang Province, China (NO.2018C01064), and Zhejiang Natural Science Foundation, China (No. LY19F020034).

REFERENCES

- [1] J. Cheng, C. Liu, M. Zhou, Q. Zeng, and A. Ylä-Jääski, "Automatic composition of semantic web services based on fuzzy predicate petri nets," *IEEE Transactions on Automation Science and Engineering*, 12(2), pp. 680-689. 2015.
- [2] E. Khanfir, R. B. Djemaa and I. Amous, "Automatic Adaptable Intentional Service Generating and Publishing Framework using OWL-S Annotation," *International Journal of Web Services Research (IJWSR)*, 15(1), pp. 1-26. 2018.
- [3] A. Urbieto, A. González-Beltrán, S. B. Mokhtar, M. A. Hossain, and L. Capra, "Adaptive and context-aware service composition for IoT-based smart cities," *Future Generation Computer Systems*, 76, pp. 262-274. 2017.
- [4] Z. Xiong, W. Luo, L. Chen, and L. M. Ni, "Data vitalization: a new paradigm for large-scale dataset analysis," In *2010 IEEE 16th International Conference on Parallel and Distributed Systems*. IEEE. 2010, pp. 251-258.
- [5] W. Zhou, L. Liu, C. Pu, et al, "An Experimental Study of a Biosequence Big Data Analysis Service," *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 237-244.
- [6] R. Milner, "Bipgraphical reactive systems," *International Conference on Concurrency Theory*. Springer, Berlin, Heidelberg, 2001, pp. 16-35.
- [7] L. Yu, W. T. Tsai, X. Wei, et al. "Modeling and analysis of mobile cloud computing based on bigraph theory," *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. IEEE, 2014, pp. 67-76.
- [8] M. Calder, A. Kolioussis, M. Sevegnani and J. Sventek, "Real-time verification of wireless home networks using bigraphs with sharing," *Science of Computer Programming*, 80, pp. 288-310. 2014.
- [9] J. Zeng, H. L. Sun, X. D. Liu, T. Deng and J. P. Huai, "Dynamic evolution mechanism for trustworthy software based on service composition," *Journal of Software*, 21(2), pp. 261-276. 2010.
- [10] H. Huang, X. Chen and Z. Wang, "Failure recovery in distributed model composition with intelligent assistance," *Information Systems Frontiers*, 17(3), pp. 673-689. 2015.
- [11] W. Ettazi, H. Hafiddi, M. Nassar, and S. Ebersold, "Micats: Middleware for context-aware transactional services," In *International Conference on Enterprise Information Systems*, Springer. 2015, pp. 496-512.
- [12] M. Asim, A. Yautsiukhin, A. D. Brucker, et al, "Security policy monitoring of BPMN - based service compositions," *Journal of Software: Evolution and Process*, 30(9), e1944. 2018.
- [13] R. Milner, "Axioms for bigraphical structure," *Mathematical Structures in Computer Science*, 15(6), pp. 1005-1032. 2005.
- [14] N. Trčka, W. M. P. Van der Aalst and N. Sidorova, "Data-flow anti-patterns: Discovering data-flow errors in workflows," *International Conference on Advanced Information Systems Engineering*. Springer, Berlin, Heidelberg, 2009, pp. 425-439.
- [15] X. Liu, Y. Ma and G. Huang et al, "Data-driven composition for service-oriented situational web applications," *IEEE Transactions on Services Computing*, 8(1), pp. 2-16. 2015.
- [16] G. H. Alferez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz, "Dynamic adaptation of service compositions with variability models," *Journal of Systems and Software*, 91, pp. 24-47. 2014.
- [17] M. Bashari, E. Bagheri, W. Du, "Self-healing in service mashups through feature adaptation," *Proceedings of the 21st International Systems and Software Product Line Conference*, ACM. 2017, pp. 94-103.
- [18] H. Huang, X. Chen, Z. Wang, "Failure recovery in distributed model composition with intelligent assistance," *Information Systems Frontiers*, 17(3), pp. 673-689. 2015.
- [19] S. Subramanian, P. Thiran, N. C. Narendran, et al. "On the enhancement of bpel engines for self-healing composite web services," *International Symposium on Applications and the Internet*. IEEE, 2008, pp. 33-39.
- [20] S. Rinderle, M. Reichert and P. Dadam, "Correctness criteria for dynamic changes in workflow systems—a survey," *Data & Knowledge Engineering*, 50(1), pp. 9-34. 2004.

Learning - based Adaptation Framework for Elastic Software Systems

Yingcheng Sun
Case Western Reserve University
Cleveland, OH, USA
yxs489@case.edu

Xiaoshu Cai
Case Western Reserve University
Cleveland, OH, USA
xxc239@case.edu

Kenneth Loparo
Case Western Reserve University
Cleveland, OH, USA
kal4@case.edu

Abstract—Adaptation is a concern for elastic software systems. Conventional methods like Brownout try to deactivate optional computation per request after decoupling the software into different components, to lower the workload when peaks occur. However, resource-intensive components are not always easy to isolate, and some software systems are even not separable. In this paper, we propose a new paradigm that provides each core and mandatory component a corresponding alternative component, with similar function but lower resource consumption, and use reinforcement learning in a feedback loop control for the self-adaptation process. We modified the widely used benchmark RUBiS to make it weakly coupled and add alternative components. Experiments show that our framework dramatically improves both the efficiency and effectiveness of self-adaptation.

Index Terms—adaptive software, reinforcement learning, programming paradigm

I. INTRODUCTION

Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner. Self-adaptation is the most obvious characteristic of elastic software systems that enables systems to continuously adapt themselves to uncertainty in the environment. One of the challenges in such kind of systems concerns how to make adaptation to themselves at runtime dynamically in response to possible and even unexpected changes from the environment and/or user goals [1].

Different approaches have been proposed for the design of self-adaptive software, a prominent one being architecture-based adaptation [2]. For example, a paradigm called brownout [3] successfully controls the load balance by separating software components into mandatory and optional parts and adaptively activating or deactivating optional parts to manage resource usage in software systems. However, subordinate and resource-intensive components are not always easy to be isolated, and some software systems are even not separable, like single-function mobile apps with only few but highly correlated modules. We thus propose a new adaptation framework that separates software into three different types of components: mandatory, optional and alternative. The mandatory parts must be kept running all the time, such as the critical services in the system, including data-relevant services. The optional parts, on the other hand, need not be active all the time and can be deactivated temporarily to ensure

system performance in the case of flash crowds. Compared to brownout programming paradigm [2] with only mandatory and optional components, we introduce a new type of component for the system design: alternative. Each mandatory component has its corresponding “alternative” one providing the same services but use less computation resources. Some “inseparable” software systems may not have “optional” parts, but definitely can build mandatory and corresponding alternative components.

After building self-adaptive software architecture, we also need to design a feedback loop control mechanism to fulfill the self-adaptation process. Control theory was used to adaptively determine when to activate/deactivate optional features in the applications through the feedback from software and environment, but applying control theory to adapt the software behavior is a complex problem [4], due to the difficulty of accurately modeling software, to the types of requirements and their tradeoffs [5] and to the need of instrumenting software to obtain sensor measurements and actuators [6]. Techniques from statistical machine learning have shown to be effective for feedback control in autonomic computing systems [7], and learning from system running environment can lead to improvements in accurately tuning parameters that avoids slow controller reactions to significant arrival rate changes [8]. Therefore, in this paper, we propose a reinforcement learning based framework to cope with non-stationary environment and changeable user goals at runtime by learning controlling rules to find appropriate thresholds. We also designed the algorithm to compute the priority of component in an application, and modified the benchmark RUBiS to make it separable. Experiments show that our framework dramatically improves both the efficiency and effectiveness of self-adaptation.

II. RELATED WORK

To enhance the adaptation of software system, the first step is to build the elastic software architecture with low-coupling characteristic, and then design a feedback loop control mechanism to fulfill the self-adaptation process. As discussed before, it is more efficient using machine learning than control theory as the control mechanism, so we only introduce existing works on feedback loop control through machine learning to support the online planning process of self-adaptive systems.

To make an application elastic, the designer needs to build a software architecture that can decompose the act of serving a request into different parts of the application, each dealing with a different part of the response. Some functions of a software application might be skipped when necessary. Rainbow [9] is a framework using reusable infrastructure to support runtime self-adaptation of software systems. Brownout [3] is a self-adaptive paradigm that enables or disables optional parts in the system to handle unpredictable workloads. In existing articles, the optional parts are identified as contents, components, and containers. Optional web contents on servers are to be showed selectively to users to save resource usage [10]. Components-based applications deactivate optional components to manage resource utilization [11]. In containerized clouds, each service is implemented as a container, and the optional containers can be activated/deactivated based on system status [12]. Optional parts might temporarily be deactivated so that the essential functions of the system are ensured and applications avoid saturation.

Several automatic policies based on machine learning and admission control were introduced. Desmeurs et al. [8] presented an event-driven brownout technique to investigate the tradeoffs between utilization and response time for web applications. Dupont et al. [13] proposed an automatic approach to manage cloud elasticity in both infrastructure and software. The proposed method takes advantage of the dynamic selection of different strategies. Moreno et al. [14] presented a proactive approach for latency aware scheduling under uncertainty to accelerate decision time, and applied a formal model to solve the nondeterministic choices of adaptation tactics. Li et al [15] [16] designed a multi-agents model to improve the self-adaptation of meta search systems.

Some existing works have the related idea with us to use Reinforcement learning (RL) to support the online planning process of self-adaptive systems. Amoui et al [17] use reinforcement learning to support action selection in the planning process and clarify why, how, and when reinforcement learning can be beneficial for an autonomic software system. Ho et al [18] present a model-based reinforcement learning approach that maintains a model to utilize the engineering knowledge and continuously optimizes system behavior through model-based reinforcement learning. Tianqi et al [1] combines reinforcement learning with case-based reasoning to overcome the limitations of rule-based adaptation in which decisions are only made based on static rules

The limitations of the past work are (i) the priority of software component is not discussed, and (ii) no strategy to deal with the inseparable components or applications. In next sections, we will discuss our proposed solutions.

III. LEARNING - BASED ADAPTATION FRAMEWORK

In this section, we first assign priority to each software component, and then propose a reinforcement learning based framework that supports self-adaptive activation/ deactivation of software components to avoid saturations.

A. Component Priority Assignment

Since some components of a system will be deactivated when unexpected peaks come, we want to know: which component should be deactivated first, i.e. how to determine the priority order of components that are deactivated? Two metrics are used to calculate the component priority: **computational complexity** and **usage frequency**. Computational complexity refers is the amount of resources required for running a component. The more complex a component, the more computing resources it needs. Usage frequency is the number of times that a component is invoked in the software system during a time interval. The algorithm of selecting the optional component is to choose the node with lower frequency but higher complexity first, because such kind of component needs more computing resources while it is less used by users. The more frequently a component being used, the more important and valuable it is. For component i , we have:

$$p = \delta.c/f$$

where p is the popularity of component i , δ is the control factor used for scaling component i 's priority, c refers to the computational complexity and f is the usage frequency.

B. Reinforcement Learning based Adaptation Framework

Conventional elastic software architectures usually separate software components into two parts: mandatory and optional, but not all software applications can be easily decoupled, so sometimes no components can be isolated. Another issue is what if a software still cannot adapt to the environment change after adjusting the running state of optional parts? For example, what if the workload of server is still high after deactivating the optional components? To deal with the above issues, we propose a new design paradigm that prepares alternative substitute for each of the mandatory components with similar functions but less complexity. Figure 1 illustrates the structure and working process of the proposed framework.

When the user traffic is low, the mandatory software components and the optional components are activated (if any), and the alternative parts are deactivated (Figure 1a). All functions are available and the complete service of the software is offered. When the traffic increases, the workload of server rises and the response time gets longer. In this case, optional components are deactivated in the order of priority. The user experience might be degraded, but the whole software will still work well instead of saturation. Figure 1b shows that all optional components are deactivated.

If the workload of server is still high after deactivating optional components, we need to switch the service running on mandatory components to alternative components (Figure 1c). The alternative code provides similar but "lighter" services than mandatory content such as a website with static picture instead of dynamic animation. Sometimes applications may not contain components that can be grouped as "optional", like mobile apps with few functions, so alternative substitutes are necessary.

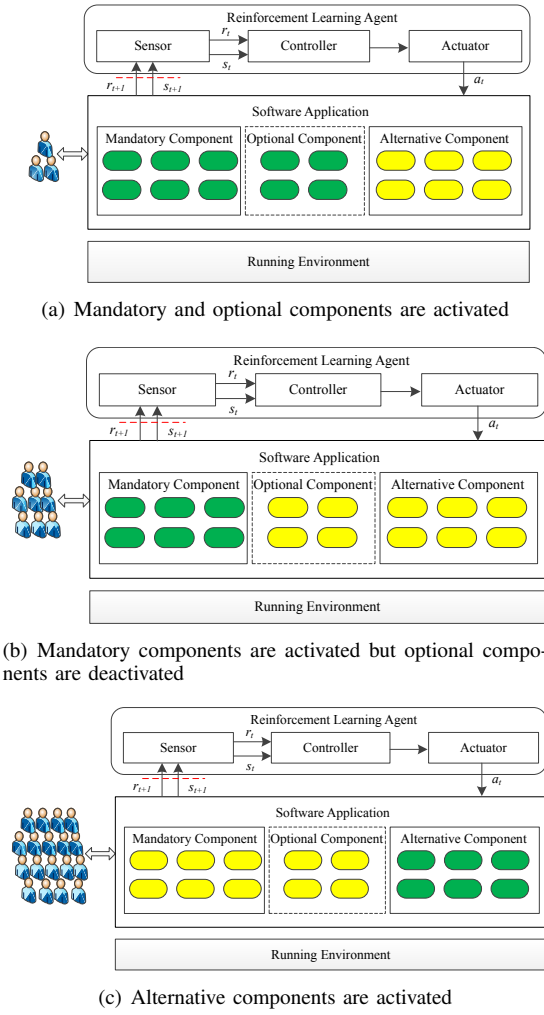


Fig. 1. The proposed reinforcement learning framework with three types of components: Mandatory, Optional and Alternative. The auto scaling process as the user traffic changes is depicted from subfigure (a) to (c)

To lower the maintenance effort, the whole feedback loop needs to be automatically managed. This would enable software applications to rapidly and robustly respond to environmental changes, and being self-adaptive. We use reinforcement learning to automatically acquire the optimal strategies as policies that controllers produce for different situations. In the reinforcement learning context, an agent takes action a_t when the system is in state s_t and leaves the system to evolve to the next state s_{t+1} and observes the reinforcement signal r_{t+1} . Decision making in elastic systems can be represented as an interaction between software controllers and environment through sensors and actuators, and the feedback reward is evaluated in the form of utility functions. An elastic system may stay in the same state, but should take different actions in different situations and workload intensity.

We study the adaptive management of resources task in which a Reinforcement Learning Agent (RLA) interacts with environment by sequentially choosing which software component to be activated or deactivated, so as to maximize

its cumulative reward. We model this problem as a Markov Decision Process (MDP), which includes a sequence of states, actions and rewards. More formally, MDP consists of a tuple of five elements (S, A, P, R, γ) as follows:

- State space S : A state $s_t \in S$ is defined as the metric function that quantifies the degree of auto-scaling variables, such as workload, response time, throughput and etc. The items in s_t are sorted in chronological order. The elasticity policy is defined in terms of rules based on the metric function: "IF workload is high AND response time is long THEN take action a_t ".
- Action space A : Each component has a running probability controlling how often it is executed, and the sum of running probability between a mandatory component and its corresponding alternative part should be 1. The action $a_t \in A$ of RLA is to change the probability of a component being activated. The action a_t is among three possible candidates:

$$a_t \in A = \{-\theta, 0, \theta\}$$

- Reward R : After the RLA taking action a_t at state s_t , i.e., changing the running probability of a component, the utility of software system changes and provides feedback, and the RLA receives immediate reward $r(s_t, a_t)$ according to the system's feedback. In this paper, we use the response time as the metric of system utility and have:

$$r_t = \lambda(resp(t) - resp(t-1))$$

where $resp(t)$ is the response time of the system at time t , and λ is a constant number scaling the reward value. If a controlling action leads to a decreased response time, the reward will increase, meaning the action is appropriate. Otherwise, if the reward is close to zero, it implies that the action is not appropriate.

- Transition probability P : Transition probability $p(s_{t+1}|s_t, a_t)$ defines the probability of state transition from s_t to s_{t+1} when RLA takes action a_t . We assume that the MDP satisfies:

$$p(s_{t+1}|s_t, a_t, \dots, s_1, a_1) = p(s_{t+1}|s_t, a_t)$$

- Discount factor γ : $\gamma \in [0, 1]$ defines the discount factor when we measure the present value of future reward. In particular, when $\gamma = 0$, RLA only considers the immediate reward. In other words, when $\gamma = 1$, all future rewards can be counted fully into that of the current action.

The upper part of each graph in Figure 1 illustrates the agent-software interactions in MDP. With the notations and definitions above, the problem of adaptive management of resources can be formally defined as follows: Given the historical MDP, i.e., (S, A, P, R, γ) , the goal is to find a controlling policy $\pi: S \rightarrow A$, which can maximize the cumulative reward for the software system. A widely-used reinforcement learning method is Q-learning that directly approximates the optimal quality function of a policy π :

$$Q^*(s, a) = \max Q^\pi(s, a)$$

and then derives the optimal policy from Q^* by selecting the highest valued action in each state:

$$\pi^* = \operatorname{argmax}_{a \in A} Q^*(s, a)$$

To approximate the optimal Q-function, Q-learning repeats the following two steps: 1) choose action a at s using policy derived from the current Q-function; and 2) take action a , and then update Q-function with the observed reward using the following formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where α is the learning rate, and γ is the discount factor that determines the present value of future rewards.

IV. EXPERIMENT

A. Experiment Setting

We verify our framework based on a widely used benchmark e-commerce web application RUBiS, with the dataset ‘‘Oops! paper dump’’ from RUBiS official website¹. The evaluation setup includes a ThinkPad laptop with Intel Core i5 2.50 GHz processor and 10 GB RAM. This web application is deployed on the laptop, while the visiting traffic is generated by a load test tool named JMeter. JMeter allows to dynamically selecting the number of users and maintains a number of client threads equal to the number of users.

RUBiS benchmark implements three functions of an auction site: selling, browsing and bidding. There are two roles in this benchmark: server for an auction and client that can be a seller, buyer or visitor. To calculate the priority of each component, we assume that one file (.class, .html or .xml) represents one unit of complexity, which means if a component includes three files of .class, its complexity is three. Though it is not very accurate to compute the running time using this method, it has little effect to our project since the complexity of each file is close to the other. Therefore, we can view each file as a unit with the same resource requirement. We use ‘‘session mechanism’’ to identify users who are visiting the website and simulate the workload with the number of visits because it is not easy to get the actual workload of the server. When the user traffic changes, controllers can switch the running components dynamically.

We select ‘‘Recommendation for You’’ as the optional component according to the sorting result of priority value. In order to make a more obvious comparison, we add website effects to the page ‘‘AboutMe’’ to construct the mandatory component and its original plain webpage without any effects is regarded as the alternative part. The website effects are implemented by CSS and JavaScript. We also add a new alternative component ‘‘ViewItem_light’’ that introduces product items in words compared to the original indivisible ‘‘ViewItem’’ module that

TABLE I
SELECTED COMPONENTS FOR EXPERIMENTS

Component Type	Function	Description
Mandatory	AboutMe	AboutMe with web effects
	ViewItem	Introduction of products in pictures
Alternative	AboutMe_light	AboutMe without web effects
	ViewItem_light	Introduction of products in words
Optional	Recommendation	Recommendation for You

presents items in pictures. All the components used in our experiment are listed in Table 1.

For JMeter, three parameters need to be set before running the sampler: number of threads, ramp-up period and the number of times to execute the test. The ramp-up time tells JMeter how long to take to run full number of threads chosen. For example, if 10 threads are used, and the ramp-up period is 100 seconds, then JMeter will take 100 seconds to get all 10 threads up and running. Listeners added to the thread group are: ‘‘aggregate report’’, ‘‘view results tree’’ and ‘‘view results in table’’. ‘‘Aggregate report’’ shows results of measurements by calling the same page lots of times as if many users are calling that page. ‘‘Result tree’’ outputs the report in whether requests are responded successfully or not. In ‘‘result table’’, ‘‘sample time’’ means the time every request uses and ‘‘latency’’ indicates the time interval between request and response from the server.

B. Evaluation

To verify the effectiveness of our proposed framework, two experiments are done in this section: running RUBiS in non-adaptive configuration and self-adaptive configuration. We first need to find the maximum capacity of our platform to acquire the configuration parameters for the ‘‘traffic peak’’. By gradually increasing the number of threads in the sampler, we obtain the lower limit of the peak condition: ‘‘launch 4000 threads in 4 seconds’’. Figure 2 shows the result.

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP request	1407	1397	1138	1612	993	4353	89.98%	313.9/sec	647.3
TOTAL	1407	1397	1138	1612	993	4353	89.98%	313.9/sec	647.3

(a) Aggregate report with errors and high average time

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Latency
1	11:38:54.828	Thread Group 1-148	HTTP request	998	🔴	1720	0
2	11:38:54.824	Thread Group 1-141	HTTP request	1002	🔴	1720	0
3	11:38:54.830	Thread Group 1-149	HTTP request	996	🔴	1720	0
4	11:38:54.897	Thread Group 1-180	HTTP request	1012	🔴	1720	0
5	11:38:54.833	Thread Group 1-152	HTTP request	1046	🔴	1720	0
6	11:38:54.865	Thread Group 1-167	HTTP request	1025	🔴	1720	0
7	11:38:54.875	Thread Group 1-188	HTTP request	1005	🔴	1720	0
8	11:38:54.839	Thread Group 1-156	HTTP request	1041	🔴	1720	0
9	11:38:54.880	Thread Group 1-169	HTTP request	1021	🔴	1720	0
10	11:38:54.845	Thread Group 1-161	HTTP request	1036	🔴	1720	0
11	11:38:54.881	Thread Group 1-175	HTTP request	1020	🔴	1720	0
12	11:38:54.828	Thread Group 1-145	HTTP request	1053	🔴	1720	0
13	11:38:54.842	Thread Group 1-158	HTTP request	1038	🔴	1720	0
14	11:38:54.868	Thread Group 1-182	HTTP request	1014	🔴	1720	0
15	11:38:54.842	Thread Group 1-159	HTTP request	1041	🔴	1720	0
16	11:38:54.873	Thread Group 1-186	HTTP request	1011	🔴	1720	0
17	11:38:54.860	Thread Group 1-174	HTTP request	1024	🔴	1720	0
18	11:38:54.824	Thread Group 1-144	HTTP request	1060	🔴	1720	0
19	11:38:54.826	Thread Group 1-146	HTTP request	1069	🔴	1720	0
20	11:38:54.869	Thread Group 1-181	HTTP request	1016	🔴	1720	0
21	11:38:54.846	Thread Group 1-162	HTTP request	1039	🔴	1720	0
22	11:38:54.868	Thread Group 1-172	HTTP request	1027	🔴	1720	0
23	11:38:54.855	Thread Group 1-168	HTTP request	1030	🔴	1720	0
24	11:38:54.881	Thread Group 1-182	HTTP request	1005	🔴	1720	0
25	11:38:54.832	Thread Group 1-151	HTTP request	1054	🔴	1720	0
26	11:38:54.868	Thread Group 1-179	HTTP request	1019	🔴	1720	0

(b) View of results with failed request

Fig. 2. Peak evaluation

Figure 2a shows that the total number of successful samples is only 1407 because the web server cannot process all requests

¹<https://rubis.ow2.org/>

given only a limited time. The maximum response time set to 4s is because a study made by Amazon shows that 25% users will leave when the responding time is over 4s. The error rate is 89.98%. Errors are reflected by warning status in Figure 2b.

Next, we make comparison experiments between non-adaptive and self-adaptive patterns on the module "About Me". It asks to register first before accessing the page, so we package username and password information as a request and send it to the server. To make it close to the real scenario, we simulate different users to visit the web page instead of one user visit multiple times. The method is to put 100 users' information extracted from database into a ".dat" file with "nickname" and "password" as the parameters and package it as a request (See Figure 3).

Fig. 3. Multi-users parameter. "Name" means the name list of variables used in the project. "Nickname" is the first (No. 0) column in test.dat and "password" is the second (No. 1) column in test.dat.

Non-adaptive Experiment: we observe the response time and error rate on three different running modes while the visits increase without adaptation mechanisms: 1) Mandatory + Optional components ("AboutMe" and "Recommendation") 2) Mandatory component only ("AboutMe") 3) Alternative component only ("AboutMe_light"). To evaluate the three modes, we set up samplers with the number of threads ranging from 250 to 3500 and launch in 4 seconds each time, and calculate the average response time and error rate for each sample. Figure 4 shows the performance of RUBiS with the three modes.

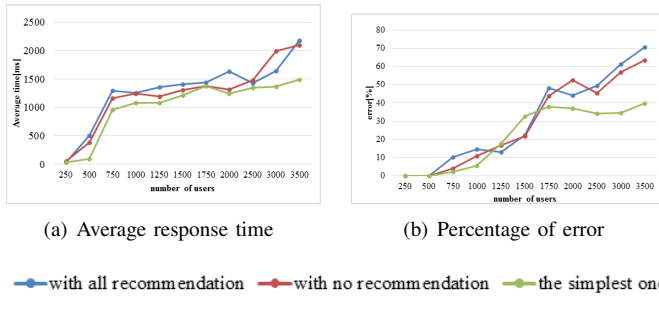


Fig. 4. Performance of RUBiS with three different modes.

Figure 4a shows that average response time increases with the number of visit users increases. It basically takes the longest time for a web server to support the functions in mode 1, because the "Recommendation" module involves frequent interactions with database. The performance improves a little after deactivating the "Recommendation" module in mode 2, and it improves obviously after switching the mode 2 to mode 3. In addition, Figure 4b shows that although some fluctuations appear in the graph, the trend is still very clear that the error

rate tends to be flat when it is approaching to saturation after switching Mandatory component into an Alternative one.

Self-Adaptive Experiment: to make the system self-adaptive, we first need to model the change of running environment. In this paper, we model the change of server workload. Since it is not easy to detect the real workload, we thus choose the response time as the metric, and the longer response time means the higher workload. However, we notice that the response time of failed requests are sometimes even shorter than successful ones in JMeter, which will interfere the whole process and make the result inaccurate. Figure 5 shows an example.

id	time	url	method	status	response	error	message
538	04:51:34.646	Thread Group 1-702	HTTP request	1026	1720	0	
539	04:51:34.580	Thread Group 1-680	HTTP request	1082	1720	0	
540	04:51:34.660	Thread Group 1-705	HTTP request	1005	1720	0	
541	04:51:32.917	Thread Group 1-34	HTTP request	4631	6285	4631	
542	04:51:32.924	Thread Group 1-38	HTTP request	4625	6285	4625	
543	04:51:32.924	Thread Group 1-40	HTTP request	4627	6285	4627	
544	04:51:32.871	Thread Group 1-16	HTTP request	4687	6285	4687	

Fig. 5. Sample time of failed requests are shorter than successful ones

Due to the above property of JMeter, we evaluate the self-adaptive experiment by launching a limited number of threads to make sure there are no failed requests and the response time is in linear growth. We set the maximum number of threads to 500 and ramp-up period to 5 seconds, and we make sure there are no errors with the number of requests less than 500. We set the control factor δ to 1 for each component and set two elasticity policies for state space: "IF response time is over 3000 ms THEN take action deactivate Optional components." and "IF response time is over 6000 ms THEN take action switch Mandatory components to Alternative parts." The whole self-adaptive process is controlled by the Reinforcement Learning Agent (RLA). Figure 6 shows the experiment results.

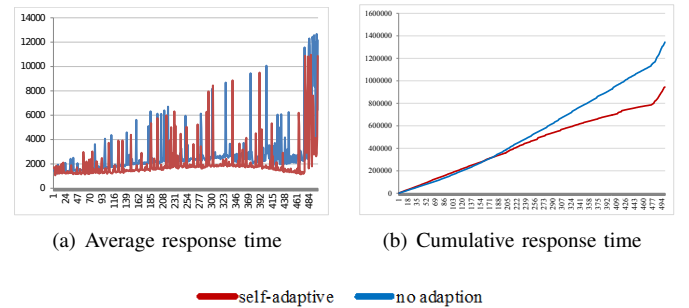


Fig. 6. Performance of running RUBiS with self-adaptive control in Mandatory, Optional and Alternative Components

When the number of visiting users is close to 150, the response time comes to 3000 ms that is our first threshold of "high workload". RLA captures the high workload through its sensor and take the action of deactivating the Optional components. However, we find that there is no obvious improvement in performance as shown in Figure 6. When the number of users is close to 200, the response time comes to 6000 ms that reaches our second threshold, so RLA takes the action of switching Mandatory components to their Alternative parts. We can see that the cumulative response time decreases

dramatically compared to the system without self-adaptation mechanism from Figure 6b.

Next, we verify the effectiveness of our framework to deal with the problem of "what if a software application cannot be easily decoupled?" When all the modules are highly correlated in a software application and cannot be easily isolated, the Brownout mechanism does not work anymore. As discussed before, we propose the method of switching the whole complex module or component into a simpler one, instead of isolating optional parts. As an example, we choose the function "ViewItem" with all its components related and cannot be decoupled. We set the maximum number of threads to 700 and ramp-up period to 5 seconds. We set an elasticity policy for the state space: "IF response time is over 35 ms THEN take action *switch Mandatory components to Alternative parts*". Figure 7 shows the experiment results.

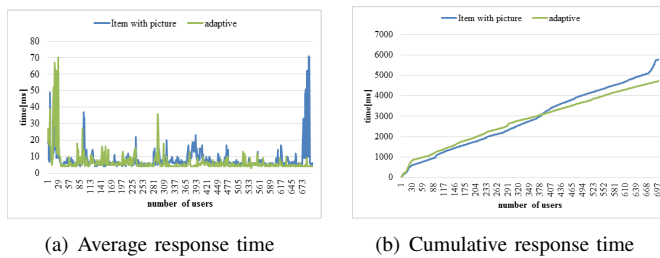


Fig. 7. Performance of running RUBiS with self-adaptive control in Mandatory and Alternative Components.

When the number of visiting users is close to 300, the response time comes to 38 MS that reaches our threshold, and then the action of switching the "ViewItem" component to its Alternative part "ViewItem_light" is taken. In Figure 7a, the response time is very high at the beginning because of the initialization of the module. After the modules switch, the response time of the system in self-adaptation mode is less than that without self-adaptation mechanism on average. From the cumulative response time shown in Figure 7b, we can see that it is very close for two curves at first, but the self-adaptive one outperforms the original one as the number of users increase.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a framework to support self-adaptation decision making. This framework separates software into three different types of components: mandatory, optional and alternative, and use reinforcement learning to design the "controller" aiming at coping with non-stationary environment and changeable user goals at runtime. It will be interesting to integrate our framework with containers to improve scheduling performance in the future.

ACKNOWLEDGMENT

This work was supported by the Ohio Department of Higher Education, the Ohio Federal Research Network and the Wright State Applied Research Corporation under award WSARC-16-00530 (C4ISR: Human-Centered Big Data).

REFERENCES

- [1] T. Zhao, W. Zhang, H. Zhao, and Z. Jin, "A reinforcement learning-based framework for the generation and evolution of adaptation rules," in *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2017, pp. 103–112.
- [2] D. Weyns, S. Malek, and J. Andersson, "Forms: Unifying reference model for formal specification of distributed self-adaptive systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 7, no. 1, p. 8, 2012.
- [3] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodríguez, "Brownout: Building more robust cloud applications," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 700–711.
- [4] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 299–310.
- [5] K. Angelopoulos, A. V. Papadopoulos, and J. Mylopoulos, "Adaptive predictive control for software systems," in *Proceedings of the 1st international workshop on control theory for software engineering*. ACM, 2015, pp. 17–21.
- [6] S. Shevtsov, M. Berekmeri, D. Weyns, and M. Maggio, "Control-theoretical software adaptation: a systematic literature review," *IEEE Transactions on Software Engineering*, vol. 44, no. 8, pp. 784–810, 2018.
- [7] M. Maggio, H. Hoffmann, A. V. Papadopoulos, J. Panerati, M. D. Santambrogio, A. Agarwal, and A. Leva, "Comparison of decision-making strategies for self-optimization in autonomic computing systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 7, no. 4, p. 36, 2012.
- [8] D. Desmeurs, C. Klein, A. V. Papadopoulos, and J. Tordsson, "Event-driven application brownout: Reconciling high utilization and low tail response times," in *2015 International Conference on Cloud and Autonomic Computing*. IEEE, 2015, pp. 1–12.
- [9] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.
- [10] J. Dürango, M. Dellkrantz, M. Maggio, C. Klein, A. V. Papadopoulos, F. Hernández-Rodríguez, E. Elmroth, and K.-E. Årzén, "Control-theoretical load-balancing for cloud applications with brownout," in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 5320–5327.
- [11] F. Alves, G. Delaval, E. Rutten, and L. Seinturier, "Language support for modular autonomic managers in reconfigurable software components," in *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2017, pp. 271–278.
- [12] M. Xu, A. V. Dastjerdi, and R. Buyya, "Energy efficient scheduling of cloud application components with brownout," *IEEE Transactions on Sustainable Computing*, vol. 1, no. 2, pp. 40–53, 2016.
- [13] S. Dupont, J. Lejeune, F. Alves, and T. Ledoux, "Experimental analysis on autonomic strategies for cloud elasticity," in *2015 International Conference on Cloud and Autonomic Computing*. IEEE, 2015, pp. 81–92.
- [14] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: a probabilistic model checking approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 1–12.
- [15] Q. Li and Y. Sun, "An agent based intelligent meta search engine," in *International Conference on Web Information Systems and Mining*. Springer, 2012, pp. 572–579.
- [16] Q. Li, Y. Zou, and Y. Sun, "Ontology based user personalization mechanism in meta search engine," in *International Conference on Uncertainty Reasoning and Knowledge Engineering*. IEEE, 2012, pp. 230–234.
- [17] M. Amoui, M. Salehie, S. Mirarab, and L. Tahvildari, "Adaptive action selection in autonomic software using reinforcement learning," in *Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08)*. IEEE, 2008, pp. 175–181.
- [18] H. N. Ho and E. Lee, "Model-based reinforcement learning approach for planning in self-adaptive software system," in *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*. ACM, 2015, p. 103.

Self-Adaptive software changes analysis method based on “Detection-Recognition” Mechanism

He Zhang, Qingshan Li*, Lu Wang*, Wen Cheng
Department of Software Engineering
Xidian University
Xi'an, P. R. China
qshli@mail.xidian.edu.cn

Abstract—Self-Adaptive Systems (SASs) need to analyze software changes accurately and continuously, that is, recognize events caused by changes, and adjust structure or behavior. However, present event recognition methods frequently monitor events, resulting in waste of system resources. And most of them ignore the impact of operating environment uncertainty, causing errors in recognizing the event and directly affecting the reliability of SASs. Addressing the above problems, this paper proposes an event recognition method based on "detection-recognition" mechanism. Firstly, the Naive Bayesian Classification algorithm is used to detect the state of the system. If the system is judged to be abnormal, we will combine with rule reasoning and fuzzy reasoning to recognize events. The system does not have to monitor the occurrence of events from time to time, avoiding the waste of system resources. Moreover, the probabilistic reasoning method of Bayesian Classification and the introduction of fuzzy reasoning can cope with environmental uncertainty and improve the accuracy of event recognition. Finally, we exemplify this mechanism with the Web system, which proves the effectiveness of the methods.

Keywords—component; Self-Adaptive software; Analyze; Naive Bayes Classification; Rule reasoning

I. INTRODUCTION

Self-adaptive Systems (SASs) modify their behaviors or structures in response to their perception of the environment and the system itself [1]. Adaptation process for SASs generally includes monitor, analyze, plan and execution [2]. The **Analyze** is responsible for judging whether the system needs to be adjusted by observing the changes information of the Monitor, and to recognize accurately the events triggered by the changes.

At present, there are few researches on changes analysis. Some researchers apply **ontology** to analyze system situation [3][4]. The accuracy of ontology reasoning is higher, but the ontology has high demand for developers, and it is difficult to perform dynamic correction during system running, so we will not consider the ontology reasoning method. Some researchers use **rule** methods to recognize environmental events [5], and some recognize active database events [6]. The types of events recognized in the above studies are **single or limited**. Nowadays, the operating environment or system structure is highly likely to change. If the ontology or rules are **unchanged**, the **accuracy** of

event recognition will be low.

There are other methods for recognizing events, including event listeners [7], a generalized modeling framework of fault detection and correction processes [8]. Most methods lack research on system state detection, they usually recognize events directly. However, software changes do not necessarily trigger events, for example, fluctuations and surges of system status data will not affect the normal provisioning of system functions and system status, so they will not evolve into events as the system runs. Such frequent monitoring of events will frequently make use of system resources, resulting in **waste of resources**.

In addition, if the event is recognized incorrectly, even if the system performs a series of adjustments, it may not achieve the expected results, or even make the system crash, which seriously affects the **reliability** of the SASs. Nowadays, the dynamic operating environment of complex software and the complexity of its structure cause the process of event recognition faces **uncertainties** such as environmental complexity and ambiguity of demand. Most methods focus on recognizing events in a certain environment, its accuracy cannot be guaranteed in the dynamic and variable environments. The present ideas of processing uncertainty mainly include **fuzzy logic** and **probability theory** [9][10]. These methods only consider the research of design phase, or have specific scenario constraints.

In response to above issues, this paper proposes a “**detection-recognition**” mechanism, which first judge the system state, if it is abnormal, then recognize the event. In the “detection” stage, we establish the **Naive Bayesian Classification** model to judge system status quickly by analyzing the probability value. In the “recognition” stage, we combine the **rule reasoning** and **fuzzy reasoning** to recognize the event, which can improve the accuracy of event recognition and migrate this method to other systems through the addition and modification of rules.

This paper is organized as follows: section II provides the detail of our event recognition method; section III introduces our experiment and some discussions; conclusion is discussed in section IV.

II. THE EVENT RECOGNITION METHOD

We propose the "detection-recognition" mechanism to analyze events triggered by the changes. It contains two stages of abnormal state detection and event recognition.

A. Abnormal State Detection

Abnormal state definition: the system's functional or non-functional requirements are affected due to system events. To detect accurately and quickly the abnormal state of the system running, we apply the **Naive Bayesian Classification** model to judge the system status.

First, we convert the numerical data collected in the system log into character data by conversion threshold. The processed log is divided into the training set and the test set by the 4-fold cross-validation method.

Second, the frequency of occurrence of features or categories in the training set is counted to estimate the probability of occurrence. We adopt the *Laplace transformation method* to avoid the situation where the probability value is 0.

Then, we get the error rate of the model by operating the test set. If the error rate is higher than the preset tolerance, we will return to data processing process, and dynamically adjust the threshold of the converted numerical data. In general, the tolerance is set to 8%, this value will be verified in Section III.

Finally, the model calculates the system state, as in (1).

$$P(Category | FeatureValue) = \frac{(\prod_{i=1}^n P(FeatureValue_i | Category)) \times P(Category)}{\prod_{i=1}^n P(FeatureValue_i)} \quad (1)$$

As in (1), *FeatureValue* refers to the eigenvalues that can characterize the state of the system. *Category* is divided into *Normal* and *Abnormal* categories in this paper. When $P(Abnormal|FeatureValue)$ is greater than $P(Normal|FeatureValue)$, we consider that the current system is in an abnormal state, and further need to inference the event.

B. Event Recognition

We combine the rule-based reasoning and fuzzy reasoning to recognize events. The working process is shown in Fig.1. We first establish the events library. Then, according to the predefined recognition rule base, the event information that occurs is reasoned. At the same time, we apply fuzzy reasoning to supplement rule reasoning, further achieve feedback and correction of the rule base and ensure the accuracy of event recognition.

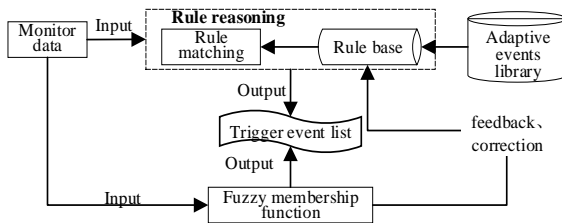


Fig.1. Event recognition method working process

1) Event recognition method based on rule reasoning

The process of this method is as follows. First, we define the corresponding mapping rules for the event. The rules indicate the relationship between system status and events, as in (2).

rule RuleName

when Judging condition , **then** Event information (&& action)
end (2)

We introduce the mapping rules between system states data and "Unit Fault" as an example, as shown in Fig.2.

```

rule UnitFault
  when SystemState (heartBeatInterval>threshold&&
    responseTime>threshold&&errorRate>threshold
    &&nodeState=="normal" )
  then <E0201> && getTime();
end

```

Fig.2. UnitFault rule example

Among them, the rule conditions are the judgment of the system status value, and the latter part of the rule is the event ID and the time that successfully matches the rule.

Then, we recognize events based on the mapping rules. We match the status information with the conditions of the rules in the rule base. If the current status information can match multiple rules, the rules will be placed in the conflict set. Conflict resolution strategies such as predefined rule priorities or definition rule groups are used to resolve conflicts between rules in conflict sets. Once the conflicts are complete, the rules will be executed in order. Then we will output event information or perform corresponding actions.

This method belongs to the category of precise matching, so events that have occurred can be accurately inferred according to the rules.

2)Event recognition method based on fuzzy reasoning

In this method, we establish fuzzy sets and membership function for the state eigenvalues of the system. Then, we establish fuzzy rules. In the rule base, the rules include the form of the fuzzy set in addition to the above-mentioned form of passing the threshold. Finally, the matching degree between the current system state and each rule is calculated by (3). We select a rule with the largest matching degree, output event or perform action in the latter part of the rule.

$$MatchingDegree_R = \sum_{i=1}^n (membership_i \times weight_i) \quad (3)$$

$MatchingDegree_R$ indicates the matching degree between the system state and rule A. n indicates the number of system state eigenvalues. $membership_i$ indicates that the eigenvalues i belongs to the membership of the fuzzy set of eigenvalues in rule R, and $weight_i$ indicates the weight corresponding to the i -th eigenvalues.

In summary, the event recognition method based on fuzzy reasoning mainly supplements and corrects rule reasoning to improve the accuracy of event recognition.

III. EXPERIMENT

To validate the methods of this paper, we choose *BookStore* System as the case to test the ability and accuracy of "detection-recognition" mechanism.

A. Bookstore system

BookStore is a e-commerce system that uses the B/S architecture to provide users with functions such as registration login, product browsing, product payment and so on. Various types of events such as server corruption, response timeout, network bandwidth change, etc. may occur during system running. And the user requirements, computing resources, system overhead, etc. in the system are easily affected by the open environment, it is not possible to define recognition rules for all events during the design phase. Therefore, *BookStore* can be used to test the ability of this method to recognize multiple event types and uncertain event.

B. The experiment for recognizing events

1) **Model system status.** When detecting the system status, we use the three characteristic values of **response time**, **page error rate** and **load** to characterize the system status. The node load is calculated by (4).

$$nodeLoad = 0.4 * CPU + 0.3 * memory + 0.3 * disk \quad (4)$$

2) **Establish event library.** We use tuples to represent event and store it in the event library to facilitate event information output during subsequent event recognition, as in (5).

$$Event = \{ E_Id, E_Name, E_Value, E_Time, E_Effect, E_Priority, E_Duration \} \quad (5)$$

E_Id is composed of 4 digits. The first two digits indicate the event type. "01" refers to the type of node resource changes. "02" refers to the type of unit resource changes such. "03" refers to the type of changes in the business logic layer. "04" refers to the type of communication environment changes. "05" refers to the type of hardware environment changes. The last two digits indicate specific events under a particular type.

E_Effect can be divided into local and global categories. The local effect refers to an event that affects only one software unit, and the global effect refers to an event that affects the global system. *E_Duration* indicates the duration of the event from being recognized to the current time, and it can be used as a reference to set the sequence of event processing.

3) Experiment Design and result

We continuously collect and store the running data of the Bookstore, and we set the dataset size as follows: the size of data set 1 to set 6 is 300, 500, 700, 900, 1100, 1300 data, respectively. The following tests are performed on a computer with Inter(R) Core(TM) i5-4570 processors and 8GB RAM.

We use the "detection-recognition" mechanism to detect the state of the *BookStore* system over a period of time and to recognize events, as shown in Table I.

TABLE I
THE SYSTEM STATUS AND EVENT DISPLAY OF BOOKSTORE

System status	Id	Name	Effect	Time	Priority
abnormal	0101	User server overload	global	11:28 2018-10-15	urgent
abnormal	0501	Home response timeout	global	12:05 2018-10-15	urgent
abnormal	0502	Reduce ads	local	12:43 2018-10-15	general
abnormal	0201	Product display page lost	local	13:18 2018-10-15	very urgent
abnormal	0301	Network delay	global	13:47 2018-10-15	urgent

We show the main attributes of the event. The priority refers to the urgency of the event to be processed, which is determined by the impact of the event on the functional and non-functional requirements of the system. We set the priority of events that have a large impact on functional or non-functional requirements to be *very urgent*, such as server damage events. The priority of a more influential event is set to *urgent*, such as response timeout. The priority of the less influential event is set to *general*, such as the user request to reduce the number of ads.

From Table I we can see that the mechanism of this paper can detect the state of the system and further recognize various types of events. It shows that the mechanism can effectively realize the main tasks of the analyze of the adaptive process.

To verify the time efficiency of the mechanism, we test the time when it processes data sets of different sizes, as shown in Fig.3. As the data size increases, the operation time of the mechanism increases, but the overall does not exceed 2500ms, indicating that the mechanism has higher time efficiency.

To verify the accuracy of the mechanism, we test the error rate of abnormal state detection under the aforementioned data sets, as shown in Fig.4. Meanwhile, the Accuracy, Precision and Recall of the method are verified under the aforementioned data sets, as shown in Table II.

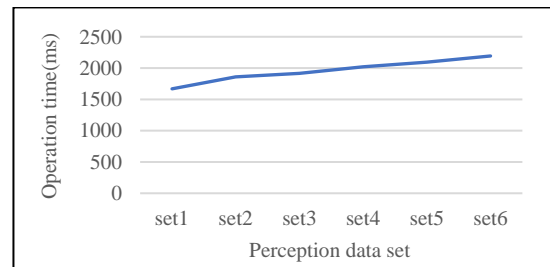


Fig.3. Operation time of the mechanism



Fig.4. Error rate of state detection method

TABLE II

ACCURATENESS OF EVENT RECOGNITION METHOD UNDER DIFFERENT DATA SIZES

Set number	Accuracy	Precision	Recall
Set 1	95.33%	99.61%	95.27%
Set 2	96.2%	99.54%	96.24%
Set 3	96.85%	99.67%	96.88%
Set 4	97.33%	99.63%	97.45%
Set 5	97.64%	99.70%	97.75%
Set 6	97.76%	99.67%	97.94%

As shown in Fig.4, as the data size increase, the error rate of the detection method is declining and gradually gradual. Since the effect of the Bayesian model depends not only on the amount of training data, but also on the construction of the classifier and the characteristics of the data to be classified, there are inevitable errors in the method for state detection. As shown in Table II, the Accuracy, Precision and Recall of the event recognition method are higher in the data sets of different scales, indicating that the method proposed in this paper has a better recognition effect.

In the state detection, to select the appropriate error rate tolerance, we set different tolerances when realizing detection method. Then we recognize events under dataset 6 to obtain the accuracy of recognition, as shown in Fig.5. The accuracy has a significant continuous decline when the tolerance is greater than 8%. At 2% to 8%, the decline is lower. If the tolerance is smaller, the system will constantly adjust and cause system overhead. Therefore, we take 8% as the tolerance.

We compare the Recall of using only rule reasoning and our method under the aforementioned data set, as shown in Fig.6. The Recall of this method is higher than that of rule-based reasoning. Fuzzy reasoning supplements the rule reasoning when encountering an unknown situation. This also verifies the effectiveness of the event recognition method in this paper.

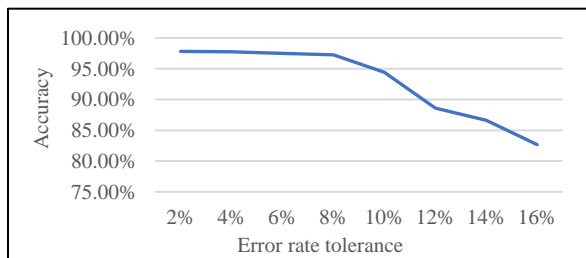


Fig.5. Impact of error rate tolerance on event recognition accuracy

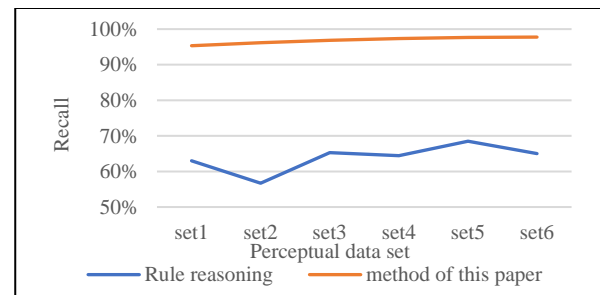


Fig.6. Comparison of event recognition methods

IV. CONCLUSION AND FUTURE WORK

In this paper, we propose a "detection-recognition mechanism, which can effectively avoid the waste of system resources, and cope with the uncertainty in the environment, so that the accuracy of event recognition is improved. In the future, we will further observe the operating characteristics of the system, and consider the online dynamic correction method of the rules. And we will expand the type of recognition event to further enhance the range of recognized events.

ACKNOWLEDGMENT

This work is supported by the Projects (61672401) supported by the National Natural Science Foundation of China; Projects (315***10101, 315**0102) supported by the Pre-Research Project of the "Thirteenth Five-Year-Plan" of China.

REFERENCES

- [1] Lemos R D, Giese H, Müller H A, et al. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap[J]. Lecture Notes in Computer Science, 2013, 5525:1-32.
- [2] Frank D, Macías-Escrivá, Rodolfo Haber, Raul del Toro, Vicente Hernandez. Self-adaptive systems: A survey of current approaches, research challenges and applications[J]. Expert Systems With Applications, 2013, 40(18).
- [3] Baader F. Ontology-Based Monitoring of Dynamic Systems[J]. 2014.
- [4] Paola A D. An Ontology-Based Autonomic System for Ambient Intelligence Scenarios[M]// Advances onto the Internet of Things. Springer International Publishing, 2014:1-17.
- [5] C. K. Chang, K. Oyama, H. Jaygarl and H. Ming, "On Distributed Run-Time Software Evolution Driven by Stakeholders of Smart Home Development (Invited Paper)," 2008 Second International Symposium on Universal Communication, Osaka, 2008, pp. 59-66.
- [6] Jin Y. Management of composite event for active database rule scheduling[C]// IEEE International Conference on Information Reuse & Integration. IEEE, 2009:300-304.
- [7] J. Lang, M. Jantošovič and I. Poláček, "Re-usability in complex event pattern monitoring," 2012 IEEE 10th International Symposium on Applied Machine Intelligence and Informatics (SAMII), Herl'any, 2012, pp. 265-270.
- [8] Okamura H, Dohi T. A Generalized Bivariate Modeling Framework of Fault Detection and Correction Processes[C]// IEEE, International Symposium on Software Reliability Engineering. IEEE Computer Society, 2017:35-45.
- [9] Yang Q, Jian Lü, Li J, et al. Toward a fuzzy control-based approach to design of self-adaptive software[M]. 2010.
- [10] Xu L, Wang X L, Wang X F. Fast Method of Compound Event Probability Calculation Based on Binary Tree[C]// Fifth International Conference on Natural Computation. IEEE Computer Society, 2009.

morph-GraphQL: GraphQL Servers Generation from R2RML Mappings (SESE)*

1st Freddy Priyatna

Ontology Engineering Group
Universidad Politecnica de Madrid
Madrid, Spain
fpriyatna@fi.upm.es

2nd David Chaves-Fraga

Ontology Engineering Group
Universidad Politecnica de Madrid
Madrid, Spain
dchaves@fi.upm.es

3rd Ahmad Alobaid

Ontology Engineering Group
Universidad Politecnica de Madrid
Madrid, Spain
aalobaid@fi.upm.es

4th Oscar Corcho

Ontology Engineering Group
Universidad Politecnica de Madrid
Madrid, Spain
ocorcho@fi.upm.es

Abstract—REST has become in the last decade the most common manner to provide web services, yet it was not originally designed to handle typical modern applications (e.g., mobile apps). GraphQL was released publicly in 2015 and since then has gained momentum as an alternative approach to REST. However, generating and maintaining GraphQL resolvers is not easy. First, a domain expert has to analyse a dataset, design the corresponding GraphQL schema and map the dataset to the schema. Then, a software engineer (e.g., GraphQL developer) implements the corresponding GraphQL resolvers in a specific programming language. In this paper we present an approach that generates GraphQL resolvers from declarative mappings specification in the W3C Recommendation R2RML, hence, can be used both by a domain expert as without the need to involve software developers to implement the resolvers, and by software developers as the initial version of the resolvers to be implemented. Our approach is implemented in morph-GraphQL.

Index Terms—GraphQL, R2RML, OBDA

I. INTRODUCTION

Introduced in 2000, Representational State Transfer (REST) has become the most common manner to provide web services in the last few years. Those web services that conform to the REST principles, known as RESTful web services, use HTTP/S and its operations to make requests to the underlying server, such as GET to retrieve objects, POST to add objects, PUT to modify objects and DELETE to remove objects, among others.

Over the years, the complexity of modern software concept has evolved since the inception of REST. For example, typical mobile applications have to take into account aspects that receive little attention in traditional applications, such as the size of data being exchanged/transmitted and the number of API calls being made. These aspects are relevant to the problem known as *over-fetching* and *under-fetching*. Over-fetching refers to the situation in which a REST endpoint returns more

data than what is required by the developer. For example, a developer may need some information about the name of a user so she hits the corresponding endpoint (`/user`). However, the endpoint may return information that is not needed by the client, such as birth date and address. The opposite also raises a problem, which is having the REST endpoint provide less data than required. Such a case is called under-fetching. It refers to the situation in which a single REST endpoint does not provide sufficient information requested by the client. For example, in order to obtain the names of all friends of a particular user, typically two endpoints may be needed: the first is the endpoint that returns the identifiers of all the friends (`/friends`), and the second is the one that returns the details of each of the friends based on the identifier (`/user`).

In order to ameliorate the aforementioned problems, Facebook proposed the GraphQL query language [6], initially being used internally by the company in 2012. GraphQL was released for public use in 2015 and since then has been adopted by companies from various sectors such as technology (GitHub), entertainment (Netflix), finance (PayPal), travel (KLM), among others. Two main components of a GraphQL server are **schema** and **resolvers**. The GraphQL schema specifies the type of an object together with the fields that can be queried. GraphQL resolvers are data extraction functions implemented in a programming language that are responsible to translate GraphQL queries into queries supported by the underlying datasets (e.g. GraphQL to SQL). GraphQL is supported by multiple GraphQL engines for major programming languages (e.g. JavaScript, Python, Java, Golang, Ruby). In addition to the above mentioned frameworks, query planning tools have been developed in order to translate GraphQL queries into other query languages (e.g. dataloader¹, joinmonster²).

¹<https://github.com/facebook/dataloader>

²<https://join-monster.readthedocs.io/en/latest/>

Generating a GraphQL server requires expertise from both domain experts and software developers. Typically, the following tasks need to be done:

- 1) A domain expert will analyse the underlying datasets, propose a unified view schema as a GraphQL schema and how the source datasets would need to be mapped into the GraphQL schema. Note that there is no standard mechanism to represent these mappings (e.g. the domain expert may use a spreadsheet, which is not necessarily easy to understand by another domain expert).
- 2) A software developer then implements those mappings as GraphQL resolvers, a process that takes significant resources. Given that the complexity of any given source code grows faster than the size of the source code, generating GraphQL resolvers is becoming more difficult even for a standard-sized dataset which typically contains more than a handful tables and hundreds of properties. This situation is even worse if the underlying dataset evolves considering that the corresponding resolvers have to be updated as well. GraphQL resolvers may not be easily understood by new developers who were not involved in the initial version thus bringing the possibility of introducing errors.

In this paper we propose the use of W3C R2RML [4] to specify the mapping rules that relate the source datasets and the GraphQL schema. The use of R2RML mappings is based on the idea that the use of a standard mapping language would facilitate better understanding of the mapping from the underlying data source and the exposed GraphQL schema. Furthermore, they also allow for better maintainability as R2RML mappings are declarative and independent from any programming language. Our main contribution in this paper is, taking the advantage that R2RML mappings are declarative, an approach to translate R2RML mappings to JavaScript-based GraphQL resolvers.

The rest of the paper is structured as follows: in section II we review R2RML and GraphQL and in section III we describe our approach on translating R2RML mappings to GraphQL resolvers. In section IV we present the queries, based on the example provided in the reference implementation³, that we use to test our implementations. Finally, related work and conclusion are presented in sections V and VI.

II. BACKGROUND

In this section we provide some background on two of the underlying technologies that we will use: GraphQL and R2RML. We use the example provided in the reference implementation based on the Star Wars movies to explain the background concepts. An overview of its schema in a tabular model and some of the data is shown in Figure 1.

A. GraphQL

GraphQL is a specification that provides a unified view for accessing heterogeneous datasets using its query language.

³<https://github.com/graphql/graphql-js/>

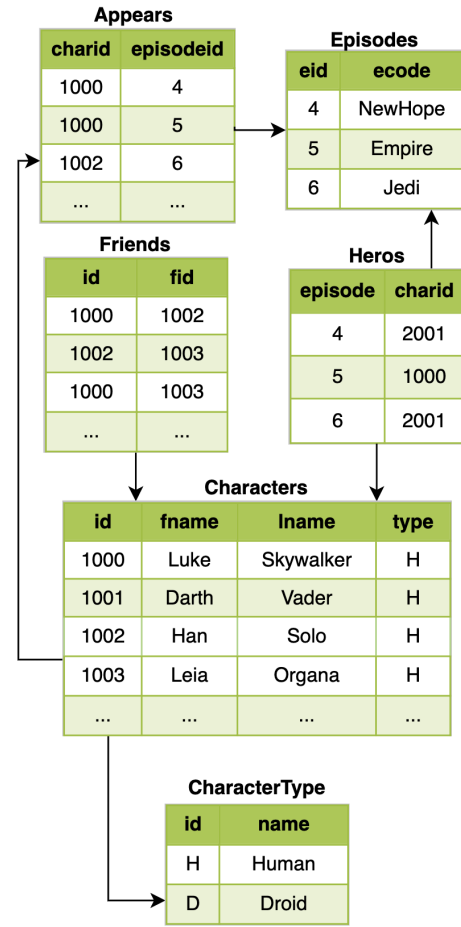


Fig. 1. Tables used in the Star Wars example, inspired by the example provided in the reference implementation

Besides the query language, the specification defines how a GraphQL server may be implemented for allowing the developers to deploy their own implementation in different programming languages. In this section we describe and provide an example of the main components of a GraphQL server: schema and resolvers (query root and type).

```

type Query {
  listEpisode(identifier:String, code:
    String): [Episode]
  ...
}

type Episode {
  identifier:String
  code:String
}

```

Listing 1. GraphQL Schema for type Episode

A GraphQL schema specifies all the available types and their properties. For example, in Listing 1 we can see that the schema for the Episode type together with its two fields: identifier and code.

A GraphQL resolver describes the relationship between the defined GraphQL types/fields and the data sources.

It implements the methods for accessing the data of each field in a specific dataset. For example, given the dataset in Figure 1, a GraphQL resolver may provide queries for retrieving all instances of the defined GraphQL types (e.g., `listAppear`, `listEpisode`, `listCharacter`, `listFriends`, `listHeroes`).

Listing 2 shows a possible JavaScript implementation of the resolver for the Episode type. This part of the code is responsible for filtering out instances based on the fields identifier or code.

```
listEpisode: function({identifier,code}) {
  let sql = `SELECT
    'ex.com/episode/' || eid AS c1
    , ecode AS c2
  FROM episodes
  WHERE
    c1 = ${identifier} AND c2 = ${code}`
  let data = db.all(sql);
  let allInstances = [];
  return data.then(rows => {
    rows.forEach((row) => {
      let instance = new Episode(
        row['c1'], row['c2']
      );
      allInstances.push(instance);
    })
  })
  return allInstances;
};
```

Listing 2. GraphQL Resolver for Type Episode

B. R2RML

The W3C R2RML Recommendation (September 2012) allows users to specify rules for transforming relational database content into an *R2RML output dataset*, the resulting graph from applying R2RML mappings. The transformation rules are defined in an R2RML mapping document that contains a set of Triples Map (`rr:TriplesMap`). Triples Maps are used to generate RDF triples from logical tables. A Triples Map consists of:

- a Logical Table (`rr:LogicalTable`) that specifies the source relational table/view.
- a Subject Map (`rr:SubjectMap`) that specifies the rule for generating the subjects of the triples.
- a set of Predicate Object Maps (`rr:PredicateObjectMap`) that consists of a pair of Predicate Map (`rr:PredicateMap`) and Object Map (`rr:ObjectMap`) that specify rules for generating predicate and object of the triples, respectively. If a join with another Triples Map is needed, a Reference Object Map (`rr:RefObjectMap`) may be specified.

A Term Map (`rr:TermMap`) is either Subject Map, Predicate Map, and Object Map. Term Maps are used to generate RDF terms, either as IRIs (`rr:IRI`), Blank Nodes (`rr:BlankNode`), or literals (`rr:Literal`). The values of the term maps can be specified using a constant-valued map (`rr:constant`), a column-valued map (`rr:column`), or a template-valued map (`rr:template`). Furthermore,

additional information such as datatype (`rr:datatype`) can also be attached to Term Maps.

In Listing 3 we show the R2RML mapping for the table Episode where the subject is defined as a template involving the eid column and a predicate-object pair involving the ecode column.

```
<TMEpisodes>
  rr:logicalTable [
    rr:table "Episodes";
  ];
  rr:subjectMap [
    rr:template "ex.com/episode/{eid}";
    rr:class schema:Episode
  ];
  rr:predicateObjectMap [
    rr:predicate schema:code;
    rr:objectMap [ rr:column "ecode" ]
  ];
.
```

Listing 3. R2RML Mapping for Episode

III. APPROACH

Our approach (Figure 2) generates GraphQL servers from R2RML mappings. Hence, mappings can be created by a domain expert in a declarative language, without the need for programming skills, while benefiting from the wide range of tools available for GraphQL in order to access data stored in tabular format (i.e., RDB or CSV). The approach consists of the following steps: 1) the generation of a SQL query, 2) the generation of schema and 3) the generation of resolvers, from each Triples Map defined in the mapping document.

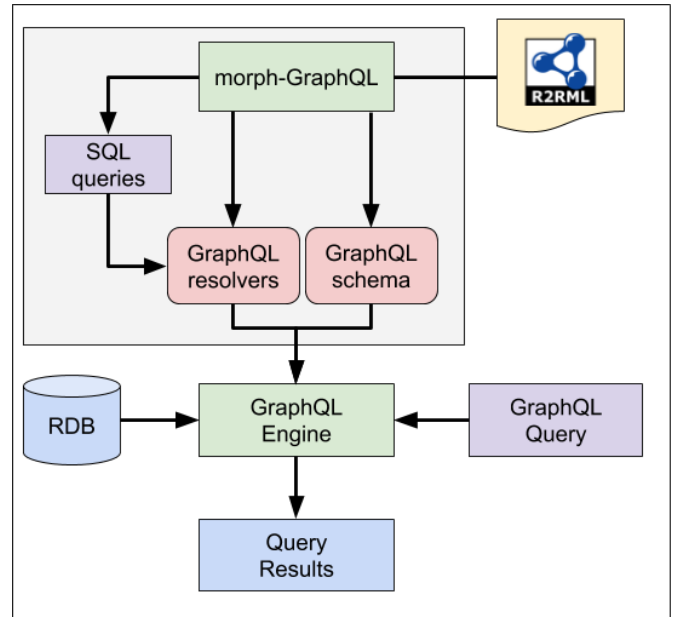


Fig. 2. **morph-GraphQL workflow.** morph-GraphQL receives R2RML mappings and generates SQL queries to be used in GraphQL resolvers. Then, it generates a GraphQL server (schema + resolvers) that can be used by a GraphQL engine to evaluate queries over the RDB data.

Auxiliary Functions. We present here a set of auxiliary functions that will be used in the functions that generate resolvers.

- *getConstant(TermMap)* retrieves the constant *c* in the constant-value term map *TermMap* = *rr:constant "c"*.
- *getColumn(TermMap)* retrieves the column *col* in the column-value term map *TermMap* = *rr:column "col"*.
- *templateToSQL(TemplateValue)* converts a template-value term map into an SQL expression. For example, given the term map *rr:template "ex.com/episode/{eid}"* as the input, this function may return *"ex.com/episode/" || eid* or *CONCAT("ex.com/episode/{eid}", eid)*, depending on the database system being used.
- *transDataType(xsdDataType)* that given an XSD Data Type return the corresponding GraphQL type. For example, *transDataType("xsd : string")* returns String.
- *join(objs, separator)* that joins a collection of objects *objs* into a string with the separator *separator*. For example, *join([1, 2, 3], "AND")* returns *"1 AND 2 AND 3"*.

A. Generating SQL Queries

We present here a set of translation functions that translates a triples map into the corresponding SQL query to be used in GraphQL resolvers. This set of functions is adapted from the work presented in [3], which is used to translate SPARQL queries into SQL queries without the presence of R2RML mappings.

- $\alpha(\text{TriplesMap})$ returns a set of logical tables associated with the triples map *TriplesMap*, which is the logical table associated to the triples map *TriplesMap* and additionally all the parent tables if *TriplesMap* contains Referenced Object Maps.
- $\beta(\text{TermMap})$ that given a term map *TermMap* returns the corresponding SQL expression, that is:
 - *getConstant(TermMap)* if *TermMap* is a constant-value map
 - *getColumn(TermMap)* if *TermMap* is a column-value map
 - *templateToSQL()* if *TermMap* is a template-value map.
- *alias(TermMap)* generates a unique alias to be used in the generation of SQL statement
- *genPRSQL(TriplesMap)* generates a SQL expression which projects the relevant SQL expressions of a triples map *TriplesMap* (i.e., β of Subject Map and all Object Maps) together with their aliases.
- *genCondSQL(TriplesMap)* generates a SQL expression which is evaluated to true if they match the arguments passed in the resolver functions and additionally the join conditions if *TriplesMap* contains Referenced Object Maps.

- finally, $\text{trans}(TM) = \text{"SELECT genPRSQL}(TM) \text{ FROM } \alpha(TM) \text{ WHERE genCondSQL}(TM) \text{"}$ translates a triples map into the corresponding SQL query.

Example Given Listing 3 as the input, *trans* generates the SQL query that can be seen in variable *sql* in Listing 2.

B. Generating Schema

Algorithm 1 generates a GraphQL schema from a Triple Map. It simply generates a GraphQL type *MappedClass*, where *MappedClass* is the class specified in the Subject Map of the Triples Map. The fields of the *MappedClass* are identifier and all the mapped predicates in the Predicate Object Maps of the Triples Map. The datatype of the fields are the results of function *transDataType*, which returns the corresponding GraphQL type from the datatype specified in the Object Maps of the Triples Map.

Algorithm 1 GenerateSchema(TriplesMap)

```

SM = TriplesMap.getSubjectMap()
MappedClass = SM.getMappedClass()
POMS = TriplesMap.getPredicateObjectMaps()
Result = "type MappedClass {"
Result += "identifier:String"
for all POM ← POMS do
    PM = POM.getPredicateMap()
    OM = POM.getObjectMap()
    PMConstant = PM.getConstant()
    DataType = transDataType(OM.getDataType())
    Result += "PMConstant:Datatype"
end for
Result += "}"
return Result

```

Example Given Listing 3 as the input, Algorithm 1 generates GraphQL Type *Episode* that can be seen in Listing 1.

C. Generating Resolvers

Algorithm 2 generates a GraphQL resolver from a TriplesMap. As for the name of the resolver, we opt for *listMappedClass*, that is, a Triples Map whose mapped class is *Episode* will generate a resolver *listEpisode*. This resolver will use the SQL query generated from section III-A, execute the SQL query on the underlying database engine, and then generate the corresponding instances by calling the constructor of Type *MappedClass*.

Example Given Listing 3 as the input, Algorithm 2 generates resolvers that can be seen in Listing 2.

IV. IMPLEMENTATION AND QUERIES

As of the time of writing, we have implemented **morph-GraphQL**⁴, an open source tool to translate R2RML mappings into Javascript-based GraphQL resolvers. Currently, it

⁴<https://github.com/oeg-upm/morph-graphql>, deployed at <http://graphql.morph.oeg-upm.net>

Algorithm 2 GenerateQueryRoot(TriplesMap)

```

SM = TriplesMap.getSubjectMap()
MappedClass = SM.getMappedClass()
POMS = TriplesMap.getPredicateObjectMaps()
PMSConstants = [identifier]
for all POM ← POMS do
    PM = POM.getPredicateMap()
    PMConstant = PM.getConstant()
    PMSConstants.push(PMConstant)
end for
Result = ""
Result += "listMappedClass:
functions(PMSConstants.join(", ")) { "
Result += "sql = trans(TriplesMap)"
Result += "rows = db.all(sql) "
Result += "allInstances = []"
for all row ← rows do
    args = []
    for all POM ← POMS do
        PM ← POM.getPredicateMapping()
        PMConstant ← PM.getConstant()
        args.push(row.[alias(PMConstant)])
    end for
    Result += "instance = new
MappedClass(args) "
    Result += "allInstances.push(instance) "
    Result += "return allInstances"
end for
Result += "}"
return Result

```

is able to generate resolvers for accessing tabular datasets, such as RDB or CSV files. We use the JoinMonster library⁵ to generate efficient SQL queries when joins are needed.

Due to the recent emergence of GraphQL, and as far as we are aware of, there has not been any standard benchmark or test-case proposed for evaluating the conformance and performance of a GraphQL-compliant framework. In order to test our approach, we use a set of GraphQL queries with various degrees of complexity proposed in the example of the reference implementation. First, we serialize the Star Wars instance data in a tabular format (Figure 1) and then generate its corresponding R2RML mapping document. Then we evaluate the queries in Table I. All the information about the dataset, mapping and queries and their results is available online⁶. Additionally, a GraphQL server that is ready to answer those queries has been also deployed⁷.

The initial version of morph-GraphQL presented in this paper shows that an R2RML mapping document can be used to generate automatically GraphQL resolvers. Besides, the implementation of the resolvers is able to cover various levels

TABLE I
STAR WARS QUERIES

No	Description	Tables Involved
Q1	Query the hero of every episode	heroes, episodes, characters
Q2	Query for the id and friends of R2-D2	characters, friends
Q3	Query for Luke Skywalker directly, using his ID	characters
Q4	Query for both Luke and Leia	characters
Q5	Verify that R2-D2 is a droid	characters, types
Q6	Verify that the hero of episode Empire is a human	heroes, characters, types, episodes

of query complexity so that it can be used as a tool for accessing heterogenous datasets via GraphQL queries.

V. RELATED WORK

Several works are at the intersection of GraphQL and Ontology-Based Data Access (OBDA) [9]. In OBDA, ontologies are used as a global view over heterogeneous local datasets and the relationship between them is specified by mappings. R2RML is an example of declarative OBDA mappings whose focus is the generation of ontology instances from relational databases. Other related declarative proposals are: RML [5] (to deal with CSVs, JSON and XML data sources), xR2RML [8] (to deal with MongoDB), KR2RML [12] (to deal with nested data) or RMLC-Iterator (for statistical CSV files) [2]. Two techniques for answering queries over the global schema are: data translation and query translation. In data-translation, a set of mapping rules is used to generate the instances of the global schema and then those instances are materialised in a triple store so that queries posed over the global schema can be evaluated by the triple store. In query-translation, queries over the global schema are translated into queries over the local schema, taking into account the information provided in the mappings, thus eliminating the need of materialisation.

The GraphQL-LD specification is proposed in [14], where the authors include a context to GraphQL queries, similar as it is proposed in JSON-LD [13]. The goal of this work is to translate GraphQL queries to SPARQL queries for querying RDF interfaces and provide a more friendly interface for the developers. Ontop [1] proposed several semantic optimisation techniques to generate efficient SQL queries resulting from the translation of SPARQL queries taking into account R2RML mappings. morph-RDB [10] presented an R2RML-based SPARQL to SQL query translation based on the approached proposed by Chebotko et. al [3]. The approach that we proposed in the paper can be considered as a query translation technique as it allows the answering of GraphQL queries over local datasets without materialising them, by translating R2RML mappings into GraphQL resolvers and delegate the query evaluation to GraphQL engines. Note, however, unlike previous approaches that take a query as their input in their run-time, morph-GraphQL is compile-time, in sense that the generation of SQL and GraphQL resolvers

⁵<https://join-monster.readthedocs.io>

⁶<https://github.com/oeg-upm/morph-graphql/wiki/Example-Star-Wars>

⁷<http://starwars.graphql.oeg-upm.net/graphql>

TABLE II
SUMMARY OF APPROACHES

Proposal	Input	Output	Type
Chebotko et al	SPARQL	SQL	run time
morph-RDB	SPARQL + R2RML	SQL	run time
ontop	SPARQL + R2RML	SQL	run time
GraphQL-LD	GraphQL	SPARQL	run time
morph-GraphQL	R2RML	SQL + GraphQL (Schema & Resolvers)	compile time

are only executed once. We summarise the aforementioned approaches in Table II.

Another relevant work is [7], in which the authors analyse and formalise the semantics and the complexity of GraphQL. Their theoretical study can be used for further analysis of the query language while their technical contributions help GraphQL developers to implement more robust interfaces for the web.

VI. CONCLUSION

In this paper we have presented an approach to generate GraphQL resolvers from R2RML mappings together with its corresponding implementation, morph-GraphQL. Note that we do not aim to replace the traditional approach of generating GraphQL schema/resolvers manually, but we position this approach as supplementary approach. This is to say, this approach allows domain experts to use the generated schema and resolvers as the initial proof of concept that can be used to query datasets without the need for software engineers to develop a full-fledged GraphQL server. Software engineers may also benefit from our approach as they may also use morph-GraphQL to generate the initial version of a GraphQL server instead of building it from scratch. In the future, we plan to support more programming languages (e.g. Java) and more data formats (e.g. JSON) and integrate morph-GraphQL with Mappingpedia [11], a repository for R2RML mappings. We also plan to evaluate our approach comparing the time taken by a domain expert to generate R2RML mappings and a software engineer programming a GraphQL resolver.

ACKNOWLEDGMENT

We are thankful to Nandana Mihindukulasooriya, Anastasia Dimou, Ben de Meester and Pieter Heyvaert, who helped us in the identifying the main contributions of our approach. The work presented in this paper is supported by the Spanish Ministerio de Economía, Industria y Competitividad and EU FEDER funds under the DATOS 4.0: RETOS Y SOLUCIONES - UPM Spanish national project (TIN2016-78011-C4-4-R) and by an FPI grant (BES-2017-082511).

REFERENCES

- [1] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering sparql queries over relational databases. *Semantic Web*, 8(3):471–487, 2017.
- [2] David Chaves-Fraga, Freddy Priyatna, Idafen Perez-Santana, and Oscar Corcho. Virtual statistics knowledge graph generation from CSV files. In *Emerging Topics in Semantic Technologies: ISWC 2018 Satellite Events*, volume 36 of *Studies on the Semantic Web*, pages 235–244. IOS Press, 2018.
- [3] Artem Chebotko, Shiyong Lu, and Farshad Fotouhi. Semantics preserving SPARQL-to-SQL translation. *Data & Knowledge Engineering*, 68(10):973–1000, 2009.
- [4] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. <https://www.w3.org/TR/r2rml/>. Accessed: 2018-12-07.
- [5] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *LDOW*, 2014.
- [6] Facebook, Inc. GraphQL. <https://facebook.github.io/graphql/June2018/>, 2018. Accessed: 2018-12-07.
- [7] Olaf Hartig and Jorge Pérez. Semantics and complexity of GraphQL. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1155–1164. International World Wide Web Conferences Steering Committee, 2018.
- [8] Franck Michel, Loïc Djimenou, Catherine Faron-Zucker, and Johan Montagnat. Translation of relational and non-relational databases into RDF with xR2RML. In *11th International Conference on Web Information Systems and Technologies (WEBIST'15)*, pages 443–454, 2015.
- [9] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. In *Journal on data semantics X*, pages 133–173. Springer, 2008.
- [10] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In *Proceedings of the 23rd international conference on World wide web*, pages 479–490. ACM, 2014.
- [11] Freddy Priyatna, Edna Ruckhaus, Nandana Mihindukulasooriya, Óscar Corcho, and Nelson Saturno. Mappingpedia: A collaborative environment for R2RML mappings. In *European Semantic Web Conference*, pages 114–119. Springer, 2017.
- [12] Jason Slepicka, Chengye Yin, Pedro A Szekely, and Craig A Knoblock. KR2RML: An alternative interpretation of r2rml for heterogenous sources. In *COLD*, 2015.
- [13] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. Json-ld 1.0. *W3C Recommendation*, 16:41, 2014.
- [14] Ruben Taelman, Miel Vander Sande, and Ruben Verborgh. GraphQL-LD: Linked Data Querying with GraphQL. In *ISWC2018, the 17th International Semantic Web Conference*, 2018.

Semantic Rule Based Program Monitoring

Luke Tudor, Jing Sun

School of Computer Science

University of Auckland, New Zealand

Emails: ltud719@aucklanduni.ac.nz;

jing.sun@auckland.ac.nz

Hai Wang

School of Engineering and Applied Science

Aston University, United Kingdom

Email: h.wang10@aston.ac.uk

Bingyang Wei

Department of Computer Science

Texas Christian University, United States

Email: b.wei@tcu.edu

Abstract—Program monitoring aims at making sure the functionalities of the software are always correctly performed during runtime. Semantic Web provides a context enriched framework for data representation and manipulation. This paper proposed the use of ontological rules and reasoning engines to monitor the dynamic behaviours of computer systems in handling of exceptional circumstances, both positive and negative, that occur at runtime within the software processes. A prototype framework was proposed on how to integrate the rule based monitoring technique together with the targeted system. To validate the proposed solution, a light control system case study together with the Unity game engine were used to develop a simulation environment for the evaluation purpose. Compared to existing solutions, the approach outlined can provide an effective software behavioural monitoring outcome.

I. INTRODUCTION

In the past, research has been done into how software program monitors can be used to verify runtime correctness of an application by providing means of identifying errors when they occur. The goal of these monitoring systems is to provide a way to formally verify the correctness of the system in a way like automatic software testing, with the stipulation that this testing is done during the running of the system. Depending on the purpose of the program monitor, such as a debugging tool for developers, a logger or a system-wide message monitor, the exact way that these monitors are implemented changes with their purpose. These monitors can also provide many other features including precise error cause locations, methods to provide alerts about certain errors occurring or ways to recover from errors, e.g., by reverting a database to a previous state. However, these systems all have in common a way to specify what constitutes an error in the program and a way to provide feedback about that error, typically by logging or throwing an exception. Despite the potential application of program monitors in possibly increasing the overall quality of software, software program monitors are not commercially widespread and are not a part of typical software development workflow [1]. Recently, advances in the semantic web and ontologies have opened new possibilities for formal program modelling and verification. Additionally, these semantic web technologies provide powerful means for verifying constraints and conditions using rules and reasoning engines which could be useful for monitoring software program behaviour [2].

Since ontologies are a promising way to model software program data and thus monitor programs, the goal of this project is to investigate whether ontologies can be used in this way, and if they can, how best ontologies may be used for monitoring program data at runtime. Consequently, evaluation of how effective ontology-based monitoring is should be done. Because program monitors provide many different feature sets, levels of correctness and speed, amongst other factors. To determine the effectiveness of ontology-based monitoring, the best way to integrate a reasoning engine to the monitoring system should be explored, since the reasoning engine will be performing the runtime verification. Finally, the proposed system should be easy to use and more useful for developers, providing a possible alternative to conventional software testing methods.

From prior research, the monitoring approach used depends to a large extent on the intended use of the program monitor. Therefore, care must be taken to ensure that the monitor is fit for purpose. Since the intention of this project is to provide a monitor that can allow for error checking at program runtime at a single application level without developer interaction, such a system need not determine the origin of error states, nor does it require the implementation of sophisticated data recovery techniques. Additionally, although there exists within more recent research proposals for systems that integrate hardware into the monitoring of low fault-tolerant applications, to increase monitoring speed and protect against hardware faults [3], such a system would not be suitable for this purpose since the proposed system should be hardware agnostic for ease of use.

Examining past solutions, not all systems provide guaranteed fault detection. The most common cause is not mistakes in the proposed systems themselves, but in poor coverage provided by the predominantly control-flow driven monitoring techniques employed. Note that programs that have stricter runtime requirements with respect to real-time computing have more rigorous speed evaluations than those that do not, implying that for such systems, performance is of a greater concern than for systems where relatively slow human interaction comprises the bulk of the total process. Regarding the use of ontologies for modelling program data, there is evidence that such activity is possible. Other data formats such as XML are already used as intermediates for transferring data between different programs using middleware technology [4]. Since on-

technologies are designed as an expansion of such technology, with more sophisticated and standardised reasoning capabilities, it stands to reason that the functionality of semantic web and ontologies is a superset of the functionality of XML [5].

The objectives of this research are to design a system that can be used by software developers to monitor the runtime behaviour of programs and demonstrate how such a system can be used. To accomplish these objectives, the modelling of program data using ontologies needs to be explored and demonstrated for use in the ontology monitoring system. Also related to the modelling of program data is the way that rules and reasoning engines can be integrated into the system to provide the constraint checking necessary for this type of program monitoring to work. Furthermore, the proposed system should be easy to integrate into monitored programs, allow for specification of many different types of constraints on the data and be fast enough so that it could be realistically used in an actual software system.

This project aimed at achieving a usable, reliable and useful way to allow for integration of any software program into a runtime behavioural monitoring system and ways to verify correctness using many rules and constraints that can be easily and quickly changed to accommodate fluctuating requirements. Since this system focusses on providing a tool for developers and is difficult to evaluate independently of its use, a case study is proposed to test the monitoring system and to show how such monitoring can be extended to any generic program given an ontology structure. It should be noted that although the goal of this system is to simply provide a means for checking errors in programs, there exists the potential to implement some business logic in the defined rules on the ontology such that when reasoned about. These rules can make useful changes in the program data based on these functional requirements. Therefore, in the system demonstration, business logic inferences to maintain data consistency are demonstrated as they show off a superset of the potential uses of ontologies compared to the relatively simplistic logging of error states.

The rest of the paper is organised as follows. Section II presents the design of the system including software architecture decisions, technologies used, and the general methods used to integrate the program monitor with the monitored program and associated ontology. Section III presents the implementation of the system, including the construction of the monitoring system, the use of ontology and reasoners and an exploration of the case study as an example of a possible use case of the proposed system. In section IV, an evaluation of the monitoring system is discussed including the success of the testing methodology, comparison with previous program monitors, discussion of proposed system features, lessons learned and possible improvements. Finally, Section V concludes the contributions and discusses the future work.

II. SYSTEM DESIGN

The system proposed for integrating program monitoring into a piece of software is to create an instance of the program monitor within the monitored program using external APIs,

and then using this monitor instance to update and receive updates from the ontology. This approach allows for easy integration with any monitored program by simply importing the relevant library whilst providing a high degree of control to the monitored program in deciding what properties are important within the monitored program. However, for the program monitor to work, an ontology and externally defined rules must be made available to the program monitor. This can be done within the monitored program by providing the location of these two files to the program monitor instance at construction time. Since updates from the program monitor should not be polled for by the monitored program, an interrupt style listener paradigm is used so that the monitored program can register an interest with the values of properties and provide code that runs when those updates are triggered. An overall data flow of the monitoring system is shown in Figure 1.

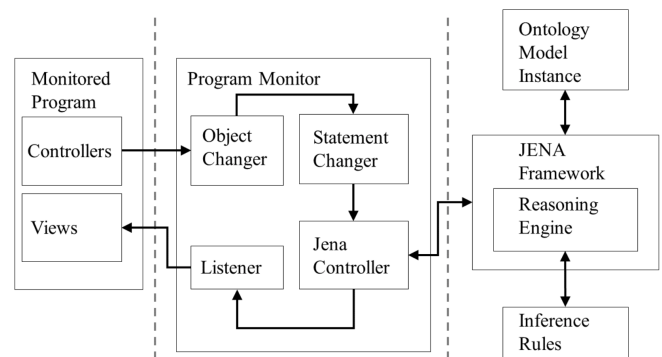


Fig. 1. Data flow through monitoring system

This call-back-like approach is known as the listener pattern and is an easy way to integrate program modules that do not need to be aware of when the other module runs given that updates are received eventually. For triggering updates, controllers are obtained from the program monitor instance to provide means to change property values from within the monitored program. The final part is the ontology model, which is updated firstly from the monitored program, then again by the reasoner if any updates are caused by the rules firing. This overall approach can be summarised as an implementation of the model-view-controller pattern (MVC) [9], which is often used for systems with a high degree of user interaction.

III. IMPLEMENTATION DETAIL

Conceptually, the monitoring system lies between the monitored program and its corresponding ontology; the monitoring itself works by checking the entire ontology against all the rules every time the Jena API is called by a property changer to update the given ontology. The timing of the rules firing is managed by the Jena framework, however, since Jena is open source, the rules engine could be made to run at different times. Since the controller objects in the monitored program use the call-back principle to specify changes, it is easy to change value types and access core Jena functionality with little additional work.

These statement changers or monitor controllers require direct access to the ontology model and must be requested from the program monitor instance rather than be constructed directly. At construction time, the full URL of the resource and property to be updated is supplied by the controller to the program monitor instance to obtain the resource and property Jena objects associated with those URLs. However, since the ontology objects for each entity change values, these property values must be located each time the object pointed to is updated. After an update is made to the ontology, all listeners are notified synchronously of the current value of the property objects that they are listening for. As with the controllers, a reference cannot be kept for a listened object, so each listener supplies URLs for the resource and property to use to retrieve listened for objects each time an update occurs. These updates are then passed to the listeners using the callback-like method, invoking a change in the monitored program somewhere. After all listeners are notified, control is passed back to the monitored program for the next update.

Before the monitoring system should be integrated into the target system, consideration must be made for the construction of the ontology and associated rules. The ontology should be constructed such that all possible conditions that might be reasonably monitored are represented in an externally logical form and such that the ontology can provide an accurate representation of the important data in the program. This means that each object in the program that represents something in the physical world should be represented in the ontology; more specifically, objects that have value outside of necessary software development usage within programs, like array lists or hash tables would only appear in the monitored program. In other words, an ontology should represent the context schema of a program. The classes of the ontology should represent the types of object to be modelled, the entities should represent the instances of those objects within the program and the ontology properties should represent the fields or attributes that are of interest within these entities. From these parts, a complete model of the ontology of a system can be constructed. For this project, OWL [5] ontology reasoning was used. Jena supports OWL DL (Description Logic) specifically, which allows more powerful reasoning than would be provided by a less expressive ontology language such as RDF. Notably, when using the proposed monitoring system, the ontology should be modified by an external tool such as Protege [10] which is designed for easy editing and analysis of ontologies.

IV. CASE STUDY AND EVALUATION

A. The Light Control System

To demonstrate the proposed system, a case study is required that demonstrates how monitored programs can be integrated with the monitoring system and how effective the proposed system is at monitoring programs. A building management system was chosen for the demonstration since building management systems are typically well-defined, contain many complex rules and constraints and are suitable for demonstration in an interactive environment such as a

game engine. Additionally, such a simulation could be feasibly extended to a physical sensor network if the associated simulation is successful at capturing all the necessary functionality in a similar way. The basis for the chosen building management system is provided in [12]. This description provides a high level of detail about a typical smart building complex with constraints related to context-aware features such as the temperature and light intensity controls [13].

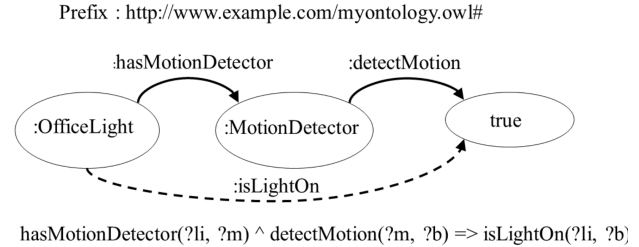


Fig. 2. Simplified ontology example.

In this example, there are at least three entities which may be formalised by an ontology, i.e., the motion detector, the light and the room which contains both these objects. Therefore, this interaction can be represented by a SWRL [11] rule which updates the ‘?light on?’ property to false whenever the motion detector in the same room has the ‘?motion detected?’ property evaluate to false. Figure 2 shows a simpler version of this scenario (without the connecting room entity) to illustrate how rules can be used to infer relationships using a reasoning engine. Extending this scenario to every room in the building with a motion detector allows the same rule to be used in each case.

B. Evaluations

For validating the monitoring system, 10 rules of varying complexity and approach were constructed to show that when updates relevant to each rule are received, that rule is fired, producing some change in the system ontology. Of these 10 rules, each was tested within the simulation environment and all rules were found to fire when expected and with the expected results. These rules were constructed such that every rule possible was paired with a rule that fired under opposite conditions and produced the opposite result. The motivation behind this rule construction methodology was to ensure that in a test environment, that the updates and rule changes were repeatable. The rules that did not have an opposite satisfied this condition of repeatability by including the object part of the triple condition within the rules as a wildcard or free variable, using this variable input to compute a variable output. This meant that some rules could contain the same functionality as two rules in the case of a Boolean literal object, or that rules which used integers could contain the same functionality as an infinite number of more specific rules. This behaviour is of interest since assertions are typically static during runtime, and different cases of assertions cannot be compressed into a single assertion. Thus, it seems that rules can sometimes provide more powerful condition checking than assertions.

For the testing environment, rooms were constructed within a simplistic mock building within the game engine so that each rule could be tested in isolation and without interference as many times as desired. This approach is a generalisation of typical testing techniques, with the main differences being that rules could be tested systematically by following a defined path through the different rooms, whilst allowing for rules to be fired at any time and with any frequency, more closely mirroring actual human interaction with a system. Although this style of testing is less automatic than traditional testing means, errors associated with timing and user experience are much easier to identify if each test is run eventually. In addition to the freeform testing provided by the simulation environment, a JUnit test suite was used early in the project lifecycle to evaluate the reasoning engine before the simulation environment had been completed. As with the simulation environment, all rules were fired when expected and produced the expected results.

A benefit of using the system is that ontologies can easily be reused and transferred between multiple formats, this contrasts with other program monitoring approaches which are more specific to each monitored program which may require more redesign work for slightly different applications. Another benefit compared to assertion-based systems is that the code is not cluttered with annotations that obscure the code intent or need extensive work to change if the monitored behaviour changes due to new requirements. This constraint checking work is delegated to the rules file, which provides a more cohesive interface for changing checked behaviour. Compared to the more common control-flow based monitoring, the proposed system is simpler to understand conceptually and easier to reason about from a monitoring perspective. This is because checking the control flow through a conditional or function often assumes some flow higher up in the control of the program, which can make it complicated to get a total view of the system status.

The proposed system can be compared to other program monitoring approaches based on each system's relative feature set. The features of several other program monitors are summarised in this paper. In comparison to systems that require specialised hardware, such as [3], this approach provides guaranteed correctness, given that the rules and ontology are constructed correctly without the hassle of customised hardware. Software only monitors to insert assertions automatically have been proposed [7], however it and other automatic assertion generating programs do not guarantee correctness unlike [3]. Systems that use constraints to monitor program execution also exist [6], however, these systems can be too heavyweight for smaller projects and do not provide the benefits of using ontologies as discussed previously. Assertion based monitors like [8] can also be used, but the main disadvantage of assertions to monitor control flow, is that control flow monitoring can become too complex to easily modify and rules must be changed from within each software module monitored. Although the monitoring system described in this paper solves the previously discussed problems, it is not

without disadvantages, the most notable being the additional work required to make and maintain the ontology and the performance of the system.

V. CONCLUSION

Program monitoring aims at ensuring functionalities of the software system are always correctly performed during run-time. This paper demonstrates not only that semantic ontology and its reasoning engines can be integrated with software applications to allow for rule-based monitoring, but also outlines a method for doing so. Additionally, the effectiveness of such a tool was evaluated with respect to how well a reasoning engine could determine errors in software and how useful the tool would be for software developers. This project has found that it is not only possible to create a powerful and flexible tool to monitor programs using rules and ontologies, but also the tool can be easily integrated with existing applications. These contributions were gathered based on implementing and testing a semi-realistic case study integrated with a game engine simulation environment to provide real-time feedback on ontology updates and rule firing. In addition, comparisons to related work were conducted with useful evaluations. In the future, a feature that would greatly increase usability would be the ability to convert rules from more common languages such as SWRL into the Jena specific rule format.

REFERENCES

- [1] A. Bertolino, *Software Testing Research: Achievements, Challenges, Dreams*, Future of Software Engineering (FOSE '07), Minneapolis, MN, 2007, pp. 85-103.
- [2] Kishore, Rajiv, Ramesh, Ram (Eds.), *ONTOLOGIES: A Handbook of Principles, Concepts and Applications in Information Systems*, Boston, MA: Springer US, 2007.
- [3] J.R. Azambuja, M. Altieri, J. Becker and F.L. Kastensmidt, *HETA: Hybrid Error-Detection Technique Using Assertions*, in IEEE Transactions on Nuclear Science, vol. 60, no. 4, pp. 2805-2812, Aug. 2013.
- [4] Steve Graham, Doug Davis, Simeon Simeonov, Glen Daniels, et al., *Building web services with Java*, Que Publishing, June 28, 2004.
- [5] W3C Recommendation 10 February 2004, *OWL Web Ontology Language Overview*, 2004.
- [6] W.N. Robinson, *Implementing Rule-Based Monitors within a Framework for Continuous Requirements Monitoring*, Proceedings of the 38th Annual Hawaii International Conference on System Sciences, Big Island, HI, USA, 2005, pp. 188a-188a.
- [7] N. Oh, P.P. Shirvani and E.J. McCluskey, *Control-flow checking by software signatures*, in IEEE Transactions on Reliability, vol. 51, no. 1, pp. 111-122, March 2002.
- [8] D. Bartetzko, C. Fischer, M. Mller and H. Wehrheim, *Jass - Java with Assertions*, Electronic Notes in Theoretical Computer Science, vol. 55, pp. 103-117, Oct. 2001.
- [9] G.E. Krasner and S.T. Pope, *A description of the model-view-controller user interface paradigm in the smalltalk-80 system*, Journal of Object Oriented Programming, vol. 1, pp. 26-49, 1988.
- [10] N.F. Noy, M. Sintek, S. Decker, M. Crubezy, R.W. Ferguson and M.A. Musen, *Creating Semantic Web contents with Protege-2000*, in IEEE Intelligent Systems, vol. 16, no. 2, pp. 60-71, March-April 2001.
- [11] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof and M. Dean, *SWRL: A semantic web rule language combining OWL and RuleML*, W3C Member Submission, vol. 21, pp. 79, 2004.
- [12] S. Queins, M. Becker, M. Kronenburg, C. Peper, R. Merz and J. Schfer, *The Light Control Case Study: Problem Description*, J.UCS: The Journal of Universal Computer Science, vol. 6, 2000.
- [13] J. Sun, H. H. Wang and H. Gu, *Semantic Enabled Sensor Network Design*, in proceedings of 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), pages 179-184, July 7-9, 2011.

Context-aware Reactive Systems based on Runtime Semantic Models

Ester Giallonardo

Dept. of Engineering, University of Sannio
estergiallonardo@gmail.com

Francesco Poggi

Dept. of Computer Science and Engineering, University of Bologna
francesco.poggi5@unibo.it

Davide Rossi

Dept. of Computer Science and Engineering, University of Bologna
daviderossi@unibo.it

Eugenio Zimeo

Dept. of Engineering, University of Sannio
eugenio.zimeo@unisannio.it

Abstract— IoT, smart cities, cyber-physical systems and sensor networks are context-aware, highly dynamic and reactive systems. Their implementation should take into account the heterogeneity of their components and make easy the management of events unplanned at design time. According to these requirements, in this paper we propose an ontology-based approach to provide runtime models of the physical entities characterizing context-aware reactive systems. We extend SSN, a W3C standard ontology, to support complex reactive behaviors through the modeling of Logical Sensors and Actuators (LSA ontology); we also present a software architecture in which a knowledge base, structured coherently with this semantic model, is bound to real world entities by grounding (via web services) semantic elements to physical sensors and actuators. To validate the approach we discuss a case study related to smart buildings for cultural heritage preservation.

Index Terms—Context modeling, Context-awareness, Semantic modeling, Semantic Sensor Networks, Ontologies

I. INTRODUCTION

Internet of Things (IoT), smart cities and cyber-physical systems propose several scenarios characterized by a high level of dynamism and heterogeneity. Applications supporting these scenarios should be context-aware since this property has been widely acknowledged as an enabler for software adaptation to dynamic changes [1]. According to [2], context is the state that a system is able to access to or modify. This state is the set of variables that are possibly shared with other systems: they can be read or modified by users, devices or applications other than the one the state is referred to.

Various recent research works take the idea of using models as central artifacts to cope with dynamic aspects of ever-changing software and its environment at runtime. Szvetits et al. [3] comprehensively survey these approaches for adaptive context-aware systems highlighting the common idea of establishing semantic relationships between executed applications and runtime models.

In this paper, we focus on the design and implementation of reactive context-aware systems taking into account the heterogeneity of the physical devices they consist of and the ability to infer high-level context properties from directly measurable

ones. Such systems need both (i) a way to describe the systems and their environments (i.e. to express architectural and state information), and (ii) a mechanism to define the application logic that drives their behaviors.

To meet the first requirement, we exploited the Semantic Sensor Network (SSN) ontology, a recent W3C recommendation [4] that has been designed to describe systems composed of a densely interconnected graph of sensors and actuators along with observations and actuations they produce. To satisfy the second requirement, we propose an extension of SSN called Logical Sensors and Actuators (LSA) ontology, since SSN does not provide mechanisms helping programmers (or reasoners) to close the gap between observations and actuations for programming context-aware reactive systems.

The LSA ontology introduces two main concepts: (software) *logical sensors* and *logical actuators*. A logical sensor (resp. actuator) is a sensor (resp. actuator) that generates observations (resp. actuations) as result of software procedures executions that use other observations as inputs. Logical sensors and actuators are entities that live only in the virtual space (e.g. knowledge base) and are connected to the external world through SSN simple sensors and actuators.

The proposed semantic runtime model is supported by a software architecture centered on a knowledge base which is bound to real world entities by grounding (mainly via web services) semantic elements to physical sensors and actuators. The behavior of the system can be specified by using sensing or actuating procedures tied to logical devices provided by the semantic model. These procedures can act upon the knowledge base by generating new facts or by redefining the structural aspects of the model thanks to the declarative approach adopted.

To clarify our approach, the overall architecture and the proposed ontology, we present a detailed scenario related to a case study in the domain of smart buildings hosting cultural heritage. The example has been implemented and tested with a prototype implementation of the proposed architecture based on Jena, OWL, and SPARQL, for the knowledge base, and RESTful services, for the interaction with the physical world, currently virtualized through an emulator.

The remainder of this paper is organized as follows. Sec-

tion II presents the related work from both research and standardization points of view. Section III introduces the SSN ontology, identifies its limitations with reference to the definition of complex and runnable sensors/actuators behaviors and proposes an extension of SSN. Section IV sketches a general architecture for context-aware applications. Section V validates the proposed ontology extension with a prototype of the infrastructure used to run an application scenario from eCulture domain. Finally, Section VI concludes the paper and highlights future work.

II. RELATED WORK

Several papers have tried to propose approaches and technologies to easily model and handle dynamic context-aware applications especially for ubiquitous and pervasive computing. One of the first ontology-based approaches is SOUPA [5]. It is expressed in OWL and includes modular component vocabularies to represent intelligent agents, time, space, events, user profiles, actions, and policies for security and privacy. However, it does not focus on sensors/actuators and reactive systems but on smart meeting places. In [6], the authors discuss the requirements that context modelling and reasoning should meet, including the modelling of a variety of context information types and their relationships, of high-level context abstractions describing real world situations, and of uncertainty of context information, without defining an ontology.

Paper [7] surveys context awareness from an IoT perspective. IoT researchers are taking into consideration Web technologies (WoTs) to support context-driven system engineering. The goal of the WoT is to extend Web services to devices, allowing a Web client to access devices properties, to request the execution of actions or to subscribe to events representing state changes [8]. The related ontology describes how to model physical or virtual sensors and actuators with the main objective of easing the binding with devices reachable through web protocols (REST, CoAP, etc.).

A different objective is pursued by the Semantic Sensor Network (SSN) ontology [4], an Open Geospatial Consortium (OGC)/World Wide Web Consortium (W3C) standard. It is mainly focused on the SOSA (Sensor, Observation, Sample, Actuator) pattern [9] to model reactive systems. It aims at supporting the definition of simple reactive behaviors that link observations, coming from modeled sensors, with the related reactions, performed by actuators. In order to link observations to physical or virtual properties, the SOSA pattern is extended with some system-oriented features. However, SSN does not directly support complex processing inside the knowledge base than asserting facts due to external sensing activities.

The Semantic Smart Sensor Network (S3N) ontology [10] is an effort that tries to specialize SSN for supporting the modeling of smart sensors. To this end a new class, `s3n:SmartSensor`, has been introduced as a specialization of `ssn:System`. A smart sensor is composed of embedded sensors, microcontrollers and communicating systems. The behavior is expressed by the execution of an algorithm (selected among the existing ones on context basis) by the

microcontroller, which can be thought as a specialization of the `ssn:Actuator`, being able to select algorithms from the current context and to change the state of the whole smart sensor. Therefore, the main purpose of S3N is to support smart sensors modeling and not to close the logical gap between sensors and actuators for fully programming reactive systems.

III. SEMANTIC MODELING OF LOGICAL SENSORS AND ACTUATORS

Semantic Sensor Network ontology: the SSN ontology was specifically designed for supporting interoperability between WoT entities taking into account performance and composition requirements. Web developers, in fact, have their concern about semantic approaches that do not assure near real time data processing. For this reason, its core module is constituted by the lightweight SOSA ontology that defines concepts and properties through schema.org annotations. The SSN main perspective is the system one.

Systems of sensors and/or actuators can be deployed on platforms for particular purposes. Actuators determine changes of the state of the world through the execution of procedures triggered by the observations of properties. SSN does not fix restrictions on the way to implement procedures, allowing to describe any information that is provided to a procedure for its use (`ssn:Input`), and any information that is reported from a procedure (`ssn:Output`). Finally, sensors detect stimuli that originated observations, i.e. events that assign results to observable properties. Stimuli can be proxies for observations of properties related to features of interest. For example, infrared sensors respond to thermal stimuli detected from the environment. The thermal stimulus is a proxy for a live presence in the sensor zone, which represents the observable property related to a feature of interest.

SSN does not allow the definition of software procedures that implement machine actionable system behaviors. To overcome this limitation, we extended SSN introducing software procedures that we mainly exploit with logical sensors and actuators, which are active, composable system components able to generate observations by processing one or more observations asserted into the knowledge base and to generate actuations in a similar way, i.e. by running actionable behaviors tied to software procedures.

Logical Sensors and Actuators ontology: Fig. 1 shows a Graffoo [11] diagram of the core elements of the Logical Sensors and Actuators (LSA) ontology², an extension of the SOSA core of the SSN ontology that allows to describe logical sensors and actuators with a specification of their behavior.

Logical sensors and actuators are modeled with the classes `lsa:LogicalSensor` and `lsa:LogicalActuator`, which are subclasses of `sosa:Sensor` and `sosa:Actuator`, respectively. The behaviors associated to logical sensors/actuators are represented by the `lsa:SoftwareProcedure` class, and the property `ssn:implementedBy` is used to connect software procedures to sensors/actuators and consequently to the logical ones.

²The Logical Sensor and Actuator ontology is available at <https://sites.google.com/site/logicalsensorsactuators>

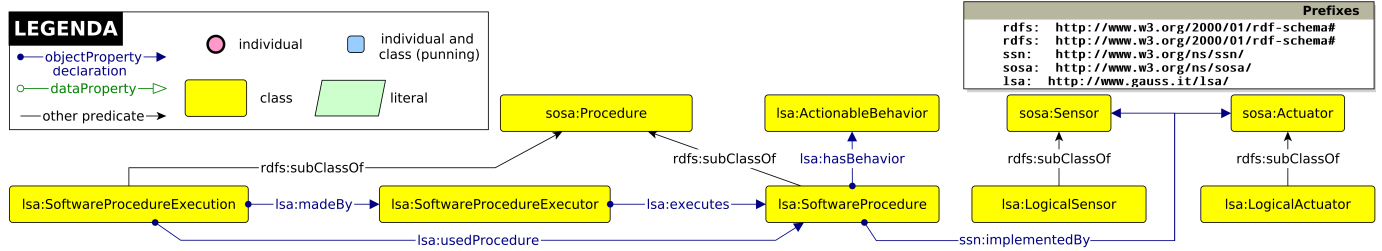


Fig. 1. Core classes of the Logical Sensors and Actuators (LSA) ontology.

A `sosa:Procedure` is defined in SSN as “a workflow, protocol, plan, algorithm, or computational method specifying how sensors make observations, or actuators make changes to the state of the world”. A `lsa:SoftwareProcedure` is a specific kind of `sosa:Procedure` with an actionable behaviour. Software Procedures may be implemented by Sensors (Actuators) or by Logical Sensors (Actuators). Sensors (Actuators) exploit software procedures for exposing how clients may interact with physical Sensors (Actuators). Through Software Procedures, a System may expose/manage its internal states or trigger an internal process. A `lsa:SoftwareProcedure` behavior is described by executable code (`lsa:hasBehavior` property).

It is important to note that the LSA ontology does not impose constraints on how such behaviors should be represented. For example, they can be modeled as OWL-S [12] processes, BPMN processes described using the BPMN Ontology [13], etc. Another key point of the LSA ontology is that it allows to discern between:

- **procedures specifications:** the algorithm, workflow, protocol, etc. used by a sensor (actuator) to perform observations (actions), along with a declaration of inputs and outputs. E.g. the algorithm used by a logical sensor that measures the perceived humidity (output) by aggregating a temperature and a humidity (input);
- **procedures executions:** the description of a specific execution of a procedure made by a sensor (actuator), which is carried out using a specific set of input values to produce a specific output. E.g. the perceived temperature X (output) of a room computed by using temperature Y and humidity Z as inputs.

In our pattern (which we aim at aligning with the ontology proposed in [14]) a procedure execution is modeled with the `lsa:SoftwareProcedureExecution` class. It is related (via the `lsa:usedProcedure` property) to a `lsa:SoftwareProcedure` and is performed (`lsa:madeBy` property) by `lsa:SoftwareProcedureExecutor`, a software agent able to execute (`lsa:executes`) a `lsa:SoftwareProcedure` that specifies the actionable behaviour (e.g. algorithm, workflow, protocol, etc.) manifested by the execution.

IV. AN ARCHITECTURE FOR SEMANTIC CONTEXT-AWARE REACTIVE SYSTEMS

We propose a reference architecture to design a framework able to host and exploit the semantic model described before for executing context-aware reactive applications. The main

component of this architecture (see Fig. 2) is the Semantic Engine. It extends a knowledge base with the machinery needed to interact with sensors and actuators and execute their software procedures. The knowledge base contains a model of the physical world it interacts with that is enriched and modified with the data coming from the sensors, assuring consistency with the physical elements it represents. This alignment is usually referred to as *causal connection*. When a modification of the model causes the enactment of an actuator to materialize this modification in the physical world we say that the model is *bi-causally connected* [15], a feature that is supported by our architecture.

To exemplify these concepts just think about a simple reactive system immersed in an environment composed by a room with a light bulb, a bulb actuator and a light sensor, all these elements are represented in a virtualized form within the system. In a causally connected system the change of the state of the real-world light bulb (turned on/turned off) is reflected in the model element that represents the bulb within the system. In a bi-causally connected system, the modification of the state of a model element is reflected as a change of state of its real-world counterpart. Thus, if we set the state of the model element representing the light bulb to off while the real-world light bulb is turned on, this triggers an actuator to turn off the bulb.

The key ingredients to actualize a system of this type are: one or more models that describe real-world conceptual classes, a binding mechanism that maps sensor observations to knowledge base updates, logical causal connections that propagate updates throughout the knowledge base, and a binding mechanism that maps updates to actuators activation for preserving the model alignment with real-world situations. It is worth noting that causal connections need some kind of computational support. According to the organization above, our architecture presents: (i) a semantic model built using the previously introduced ontologies hosted by a knowledge base platform (a triple store); (ii) a linking mechanism to report sensor readings to the system, implemented using web services exposed by the system, which is responsible of converting readings into semantic triples to insert into the triple store; (iii) a programmed logic for generating new facts from observations; (iv) an actuation mechanism exploiting actuators Web services, consistently with the WoT approach.

Causal connections are supported by rules that correlate real

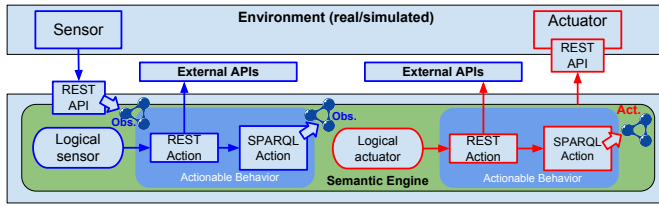


Fig. 2. Architecture outline.

world changes observed by sensors with knowledge base updates. We consistently represent these rules in the knowledge base itself: the activation part is modeled as software procedures associated to semantic sensors and actuators whereas the triggering logic is implemented by monitoring changes to the properties that are declared as inputs for these semantic sensors and actuators.

The engine (see Fig. 2) connects to the physical world by exposing a service API used to receive observations from external sensors (that can be real or simulated ones) and by invoking web service endpoints for activating external actuators or for invoking external services (on the right) to increment the capabilities of the software procedures associated to logical sensors and actuators. Whenever an external sensor notifies an observation invoking the engine's API, that observation is transformed in a semantic format and added to the knowledge base. If a logical sensor/actuator is interested in that observation (which means that it is modeled in such a way that its software procedure uses as one of its inputs the property reported by the observation) its related software procedure is executed (by running the actionable items that define the specific Actionable Behavior), producing new facts (observations or actuations) that could trigger external actuators. This approach allows for declarative definitions of reactive behaviors in a bi-causally connected system. In fact, both the model of the context and that of the system (in terms of logical sensors/actuators and their behaviors) is represented in a semantic format (e.g. by RDF triples). This allows to change the overall behavior of the system by manipulating the knowledge base: at runtime new logical sensors can be defined, the behavior of the existing ones can be modified, existing sensors/actuators can be deleted. A further advantage of this architecture is that self-adaptive behaviors can easily be implemented by simply allowing the software procedure of a sensor/actuator to work as described in [16], [17]. For example a software procedure can be activated by the detection of a failure in an external sensor to compose observations produced by other sensors in order to collect the expected events related to a feature of interest.

V. A CASE STUDY

We consider a running example derived from a larger system for Cultural Heritage preservation [18]. In a museum a new temporary exhibition is arranged. In a room of this exhibition a multimedia content has to be played. The organizers of the exhibition express the desire that the content starts playing when visitors enter the room, and stops when the room is

empty. Museum rooms have no specific detectors for knowing the number of people inside them but are equipped with Bluetooth beacons (one per room) and Infrared (IR) sensors close to the doors (used as part of the anti-theft system). Beacons notify their presence to users' personal devices equipped with a specific App turning these devices into location detectors.

We can define a logical sensor for observing the number of people in a room with two different implementations: one based on beacons and personal devices and another one based on IR sensors. If the former is faulty, the declarative approach eases the selection of another (logical) sensor able to observe the same property. In the following, we assume the first is faulty and describe the IR-based logical sensor for clarifying the overall approach and the LSA ontology.

Multimedia playback control based on a logical presence sensor:

1. a tourist crosses the door of the museum, and the two physical infrared sensors on the door sides produce two observations about the presence of a person in their detection areas;
2. a logical sensor aggregating such observations produces another observation updating the number of persons present in the rooms;
3. if the tourist enters an empty room, an actuator starts to play a multimedia flow on the room monitor; if the tourist is the last person that leaves a room before the end of the playback, an actuator will stop the multimedia flow. In both cases, the information about the new actuation is inserted into the triple store.

1. Observations made by physical sensors: Fig. 3 shows the RDF statements that are added to the triplestore by the semantic engine when a person crosses a door. Whenever this occurs, the infrared sensors placed on the two sides of the door detects the presence of a person and invokes the engine REST API in sequence (providing their ids and the instants of time when the observations occurred as request parameters).

Two observations (i.e. `gmus:observation/ir1/1` and `gmus:observation/ir2/1`) made by sensors `gmus:ir1` and `gmus:ir2` are produced, which relate to the same feature of interest (i.e. `gmus:door1`). Each observation concerns a distinct observable property (i.e. the presence in the detection area of each sensor: `gmus:presence/room1/ir1/zoneDoorInside` and `gmus:presence/room2/ir2/zoneDoorOutside`), and keeps track of the time in which the observations were performed.

In these examples we make use of punning³, an OWL metamodeling capability that allows to treat model elements as classes and individual as the same time. Elements with this double nature are represented as light blue squares in the diagram. This has been used in Fig. 3, for instance, to model the concept of infrared sensor (`gmus:IRSensor`), which is at the same time a class (i.e. a specific subclass of sensors representing infrared sensors) and an individual (since it is connected with

³See https://www.w3.org/TR/owl2-new-features/#F12:_Punning

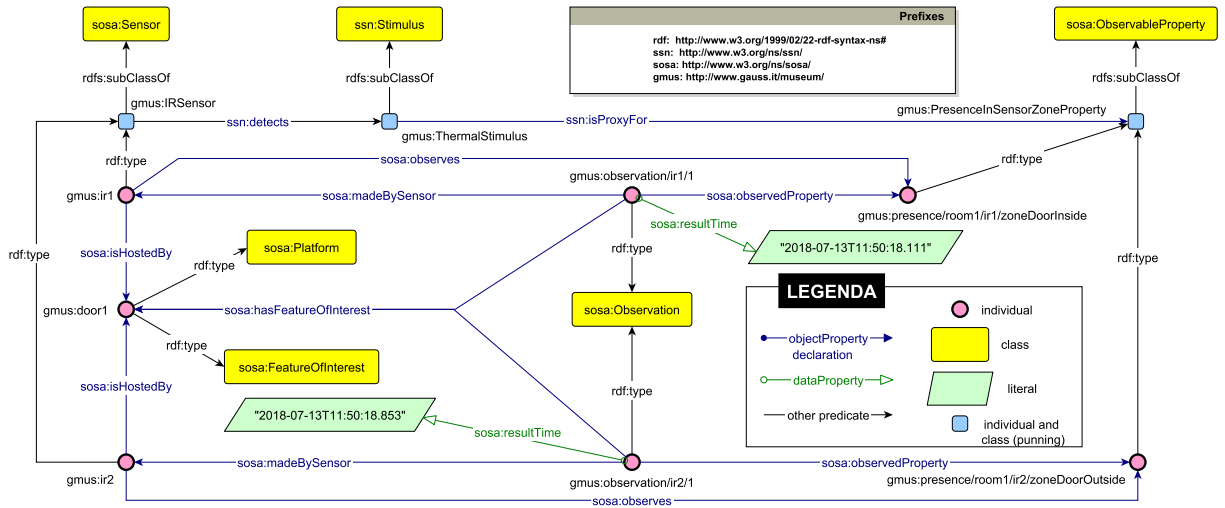


Fig. 3. Observations made by two infrared sensors.

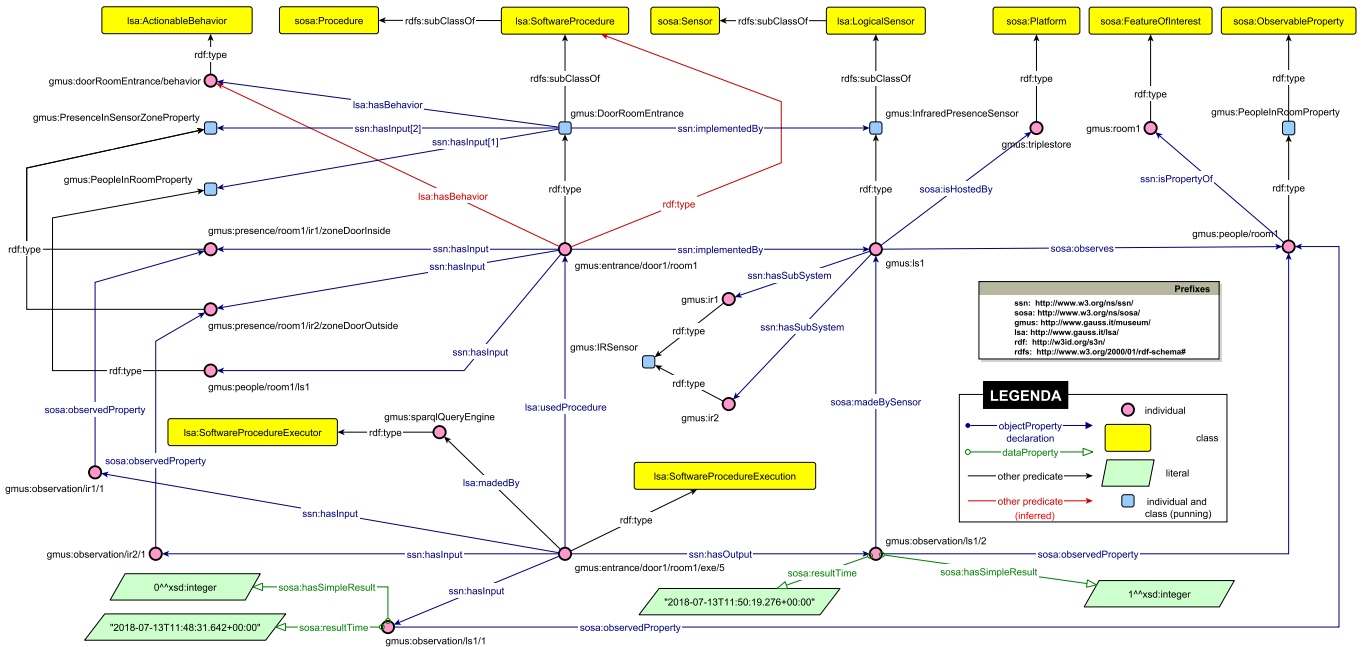


Fig. 4. Observations made by the logical presence sensor. Square brackets are used to specify property cardinality restrictions.

gmus:ThermalStimulus by the ssn:detects property). In the same way, gmus:PresenceInSensorZoneProperty is a type of observable property (i.e. subclass of sosa:ObservableProperty) and an individual (connected to gmus:ThermalStimulus by the ssn:isProxyFor property). This approach is also useful to model logical sensors behaviors, as described in the rest of this section.

2. Observations made by logical sensors: whenever a modification occurs in the triplestore (e.g. the insertion of a new observation), the semantic engine checks if one or more procedures specifying the behaviors of logical components (i.e. logical sensors and actuators) should be executed. To do so,

the engine checks if the properties related to the new observations (e.g. gmus:presence/room1/ir1/zoneDoorInside and gmus:presence/room2/ir2/zoneDoorOutside in the previous example) are specified as inputs of one or more software procedures. Since these properties (see Fig. 4) are inputs of the gmus:entrance/door1/room1 procedure (as specified by ssn:hasInput), the semantic engine identifies the procedure, which is tied to the logical sensor gmus:ls1, a specific instance of gmus:infraredPresenceSensor (the class representing logical presence sensors) hosted by the triplestore (gmus:triplestore), and executes it by observing the presence of people in the specific room (gmus:people/room1).

A mechanism is adopted by the semantic engine to retrieve behavioral information (e.g. a sequence of activities to perform) pertaining logical sensors. Since behavioral information is shared by all logical sensors of a type, the engine identifies the related software procedure (gmus:DoorRoomEntrance in our case) and retrieves the behavioral specification (gmus:doorRoomEntrance/behavior) by navigating the lsa:hasBehavior property.

Such behavioral specification in this case is composed of two actions, i.e. two SPARQL CONSTRUCT queries checking the entrance/exit in/from the room, respectively. Each of these queries retrieve the new observations made by the two infrared sensors, and if they have been performed in a short time interval - e.g. one second - produces:

- 1) a new software procedure execution (gmus:entrance/door1/room1/exe/5), connected to the software procedure (gmus:entrance/door1/room1/) by the lsa:usedProcedure property, and to the observations used as input (those made by the two infrared sensors and those pertaining the number of persons in the rooms connected by the door⁴) and the software procedure executor (gmus:sparqlQueryEngine) by the ssn:hasInput and lsa:madeBy property, respectively;
- 2) two observations as output of the procedure execution represented using the ssn:hasOutput property. For instance, the number of people in the first room has been updated from zero (in gmus:observation/lsl/1) to one (in gmus:observation/lsl/2) since a person entered the room.

3. Actuations made by logical actuators: the newly added statements (i.e. those about the observations produced by the logical sensor gmus:lsl and the relative procedure executions) trigger another control performed by the semantic engine to check logical sensors/actuators interested to those observations. In our example, the logical actuators controlling the video playback on the monitor in the room is activated, and the related software procedures is retrieved and executed, triggering the final actuation (REST invocation) of the physical device that starts the video playback on the monitor.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a proposal for an extension of the SSN ontology to support modeling of logical sensors and actuators, and their behaviors. The extension enables reactive behaviors of context-aware applications by defining the decision logic that exploits sensor observations to trigger actions. The ontology is accompanied by a an architecture that supports behaviors definition and the interaction with the real devices in the physical world. A prototype of the architecture has been implemented by using Jena, SPARQL and RESTful APIs for the interaction with the external environment, currently emulated with Freedomotic. We discussed and validated the

⁴Because of space limitations in the diagram we depicted only the observations about a room (i.e. we omitted the observations about the number of people in gmus:room2)

proposed ontology extension and the supporting architecture with the help of a case study in the domain of smart buildings for cultural heritage. The ontology extension and the related architecture represent the first step towards the definition of a more complex platform for context-awareness able to take into account failures and adaptation policies.

ACKNOWLEDGMENTS

This paper has been supported by MIUR PRIN 2015 GAUSS Project and MIUR PON VASARI Project.

REFERENCES

- [1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *International symposium on handheld and ubiquitous computing*. Springer, 1999, pp. 304–307.
- [2] A. Furno and E. Zimeo, "Context-aware composition of semantic web services," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 235–248, 2014.
- [3] M. Szvetits and U. Zdun, "Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime," *Software & Systems Modeling*, vol. 15, no. 1, pp. 31–69, 2016.
- [4] A. Haller, K. Janowicz, S. Cox, D. Le Phuoc, K. Taylor, and M. Lefrançois, "Semantic sensor network ontology," *W3C Recommendation, W3C*, 2017.
- [5] H. Chen, F. Perich, T. Finin, and A. Joshi, "Soupa: Standard ontology for ubiquitous and pervasive applications," in *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004*. IEEE, 2004, pp. 258–267.
- [6] C. Bettini, O. Brdiczka, K. Henriksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.
- [7] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 414–454, 2014.
- [8] S. Kaebisch and T. Kamiya, "Web of things (wot) thing description," *First Public Working Draft, W3C*, 2017.
- [9] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, and M. Lefrançois, "Sosa: A lightweight ontology for sensors, observations, samples, and actuators," *Journal of Web Semantics*, 2018.
- [10] S. Sagar, M. Lefrançois, I. Rebai, M. Khemaja, S. Garlatti, J. Feki, and L. Médini, "Modeling smart sensors on top of sosa/ssn and wot td with the semantic smart sensor network (s3n) modular ontology."
- [11] R. Falco, A. Gangemi, S. Peroni, D. Shotton, and F. Vitali, "Modelling owl ontologies with graffoo," in *European Semantic Web Conference*. Springer, 2014, pp. 320–325.
- [12] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne *et al.*, "Owl-s: Semantic markup for web services," *W3C member submission*, vol. 22, no. 4, 2004.
- [13] M. Rospocher, C. Ghidini, and L. Serafini, "An ontology for the business process modelling notation," in *FOIS*, 2014, pp. 133–146.
- [14] M. Lefrançois, "Planned etsi saref extensions based on the w3c&ogc sosa/ssn-compatible seas ontology paaens," in *Workshop on Semantic Interoperability and Standardization in the IoT, SIS-IoT*, 2017, p. 11p.
- [15] M. Hözl and T. Gabor, "Reasoning and learning for awareness and adaptation," in *Software Engineering for Collective Autonomic Systems*. Springer, 2015, pp. 249–290.
- [16] F. Poggi, D. Rossi, P. Ciancarini, and L. Bompani, "Semantic run-time models for self-adaptive systems: a case study," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2016 IEEE 25th International Conference on*. IEEE, 2016, pp. 50–55.
- [17] F. Poggi, D. Rossi, and P. Ciancarini, "Integrating semantic run-time models for adaptive software systems," *To appear in Journal of Web Engineering*, 2019.
- [18] E. Giallonardo, C. Sorrentino, and E. Zimeo, "Querying a complex web-based kb for cultural heritage preservation," in *Knowledge Engineering and Applications (ICKEA), 2017 2nd International Conference on*. IEEE, 2017, pp. 183–188.

Enhancing Semantic Search of Crowdsourcing IT Services using Knowledge Graph

Duankang Fu, Zhou Shufan, Beijun Shen*, Yuting Chen

School of Electronics, Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
{duankangfu, sfzhou, bjshen, chenyt}@sjtu.edu.cn

Abstract—Mining search intents in vertical websites like IT service crowdsourcing platform relies heavily on domain knowledge. Meanwhile, it still remains a difficulty of searching services in crowdsourcing platforms, as these platforms do contain much insufficient information, for example, users tend to use images describing IT services for the purpose of advertisements. To solve these problems, we build and leverage a knowledge graph to enhance searching of crowdsourcing IT services. The key idea is to (1) build an IT service knowledge graph from StackOverflow tag synonym system, Wikipedia, StuQ and data in IT service crowdsourcing platforms, (2) plug two activities into the basic search process – term expansion and service re-ranking, (3) use superordinates, hypernyms, synonyms, descriptions and relations of entities in the knowledge graph to expand user query and service information, and (4) apply a learning-to-rank model with four features to re-rank the search results, enforcing those more relevant services have the higher-ranking position. We have conducted several experiments to evaluate our approach. The results show that our approach achieves an MRR 34.9% higher and a Recall@15 11% higher than those of a basic search approach.

Keywords – IT Service Crowdsourcing; Knowledge Graph; Semantic Search; Learning-to-rank

I. INTRODUCTION

Recently, crowdsourcing has been widely used in many fields such as image recognition, taxonomy construction and entity resolution [1-2], etc. Those tasks are simple and straightforward, and people can deal with them with common knowledge. However, due to the strong professionalism and specialization, IT crowdsourcing is more complicated [3]. In a typical IT crowdsourcing platform, developers provide various types of IT services, and users search for target services according to their own requirements. The appropriate matching between user query and service information is one of the key values offered by IT crowdsourcing platforms.

Currently, almost all crowdsourcing platforms provide search function following the basic search process as shown in Figure 1. The services data are used to create the reverse index, and the user queries are segmented. And then Elasticsearch performs text matching between the queries and the reverse index. This approach adopts pure text matching technology,

which means users have to describe their requirements precisely, or, it can't identify the latent intents of users. We also find that developers tend to use images to describe their services for the purpose of advertisements in most IT crowdsourcing platforms, and thus there are not enough available textual description for services. All these lead to low performance of semantic search in IT crowdsourcing – it is difficult for users to find their target services.

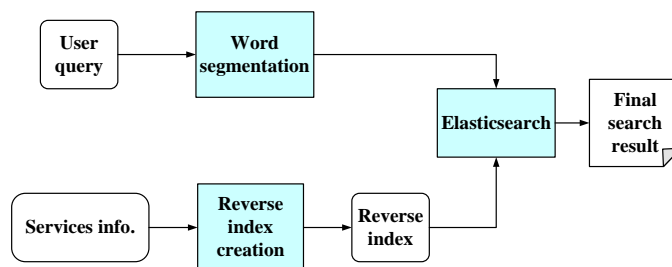


Figure 1. Basic Search Approach for Crowdsourced IT Services

As searching for crowdsourced IT services, how to understand user queries accurately? And how to complete the information of services? To address these challenges, a domain-specific knowledge graph, ITServiceKG, is constructed to enhance user query understanding. ITServiceKG mainly consists of three parts: IT service categories, IT skills and IT service instances. We insert two pluggable activities in the basic search process: term expansion and service re-ranking. After word segmentation, we use superordinates, hypernyms, synonyms, descriptions and relations of entities in ITServiceKG to expand user queries and service information. And then we get the preliminary search results from Elasticsearch, use learning-to-rank model to re-rank these results, and make the more relevant service have the higher-ranking position. We conducted several experiments to evaluate our approach. The results show that compared with the basic approach, the MRR (mean reciprocal rank) is increased by 34.871% and the Recall@15 is increased by 10.976% in our approach.

Our main contributions are summarized as follows:

1) We construct a knowledge graph of IT crowdsourcing services, which represents a complex network among IT service categories, IT skills and IT service instances.

2) We utilize ITServiceKG to expand both user queries and service information, which helps alleviate the problem that the

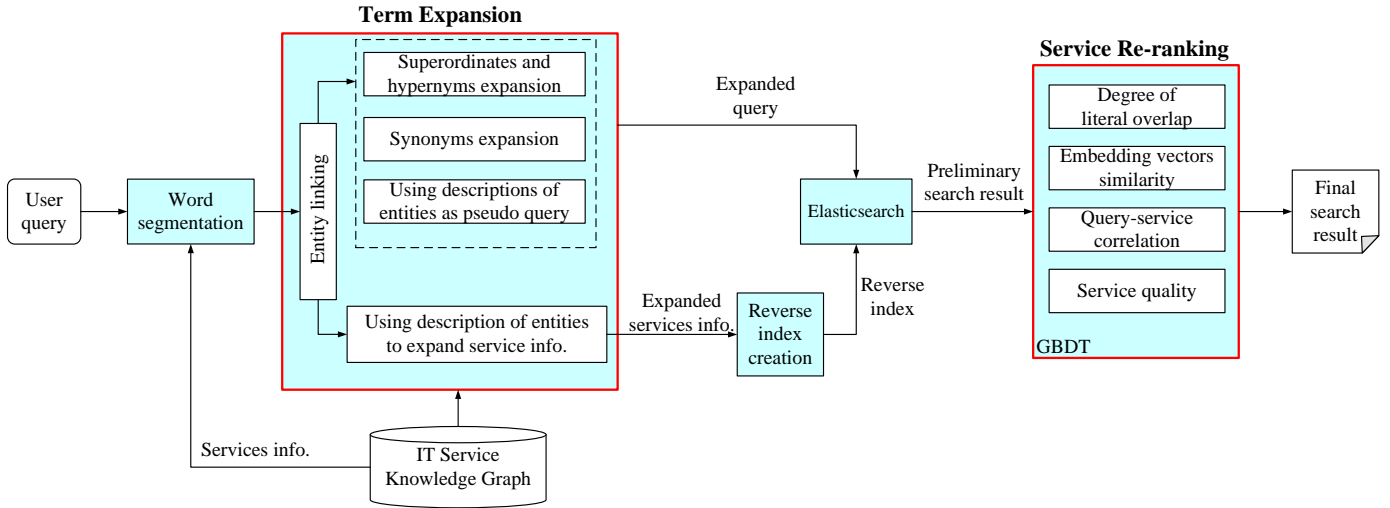


Figure 2. Overview of Our Approach

user queries are not precise and the services lack enough textual descriptions.

3) We propose a learning-to-rank model to obtain the more appropriate results. Four ranking features are designed to boost the ranking position of the more relevant services.

The rest of the paper is organized as follows. In the next section, we review some related works. The details of our approach are presented in Section III. We conduct a series of experiments to evaluate the effectiveness of our approach in Section IV. Finally, we conclude our work in Section V.

II. RELATED WORK

A. Semantic Search

Semantic search is a broad field, with many different aspects, ranging from query understanding, to answer retrieval, and result representation. In this paper, we focus on query understanding and answer retrieval. In previous research, people use query word expansion and retrieval model to obtain relevant results. They use synonyms as word expansion, extract word stems and obtain their different tenses [4], or use acronyms [5] and relevant words [6] to expand queries. The state-of-art retrieval model is learning-to-rank model applying machine learning methods, like LambdaRank, DSSM, CDSSM, etc. [7].

B. Knowledge Graph and its Application

There exists a wide range of general-purpose encyclopedic knowledge graphs, like WikiData, Freebase, DBPedia, and some domain-specific knowledge bases, such as WordNet, ConceptNet, etc. general-purpose knowledge graphs are not suitable for domain search, since these knowledge graphs are too general and could introduce redundant or unnecessary information, which leads to irrelevant search results. Some researchers construct an in-domain knowledge graph by extract related entities from a general domain knowledge graph [8]. However, this only extract a subset of a general knowledge graph, not optimized for in-domain specific purposes.

Knowledge graphs have been utilized in text information retrieval and made certain achievements [9]. Alexander Kotov, ChengXiang Zhai [10] used path finding and random walk to find related entities in ConceptNet as an expansion for queries. It mainly utilized the graph construction but neglected the text resources of entities. Jeffrey Dalton, Laura Dietz [11] proposed an Entity Query Feature Expansion (EQFE) model to make some improvement to the pseudo-relevance feedback model. Chenyan Xiong [12] proposed an EsdRank model, which treats extracted external words, terms and entities as objects in a latent space of queries and documents. Xiangling Zhang, et al. [13] proposed a concept called common semantic feature, to address the problem of entity set expansion by using KGs. Chenyan Xiong [14] proposed a word-entity duet representation model via combing traditional retrieval model and knowledge graph embedding, to describe the common features shared by the seed entities. Wen Zhang, et al. [15] proposed a new knowledge graph embedding method to learn distributed representations for entities and relations, which explicitly simulates crossover interactions. However, seldom researchers focus on the scenarios where documents lack enough text resources and the domain knowledge is not fully exploited.

III. OUR APPROACH

A. Approach Overview

To address the challenges of semantic search in IT service crowdsourcing, we propose a knowledge graph-based approach as shown in Figure 2. Our approach leverages an IT service knowledge graph (ITServiceKG) to enhance query and service understanding, and then seamlessly plugs two additional activities in the basic search process to provide precise IT service search: term expansion and service re-ranking.

- 1) *Term expansion*: After word segmentation, we apply entity linking to locate the entities of ITServiceKG in both queries and service information. Then we expand those using superordinates, hypernyms, synonyms, descriptions and relations of these entities in ITServiceKG. Thus our

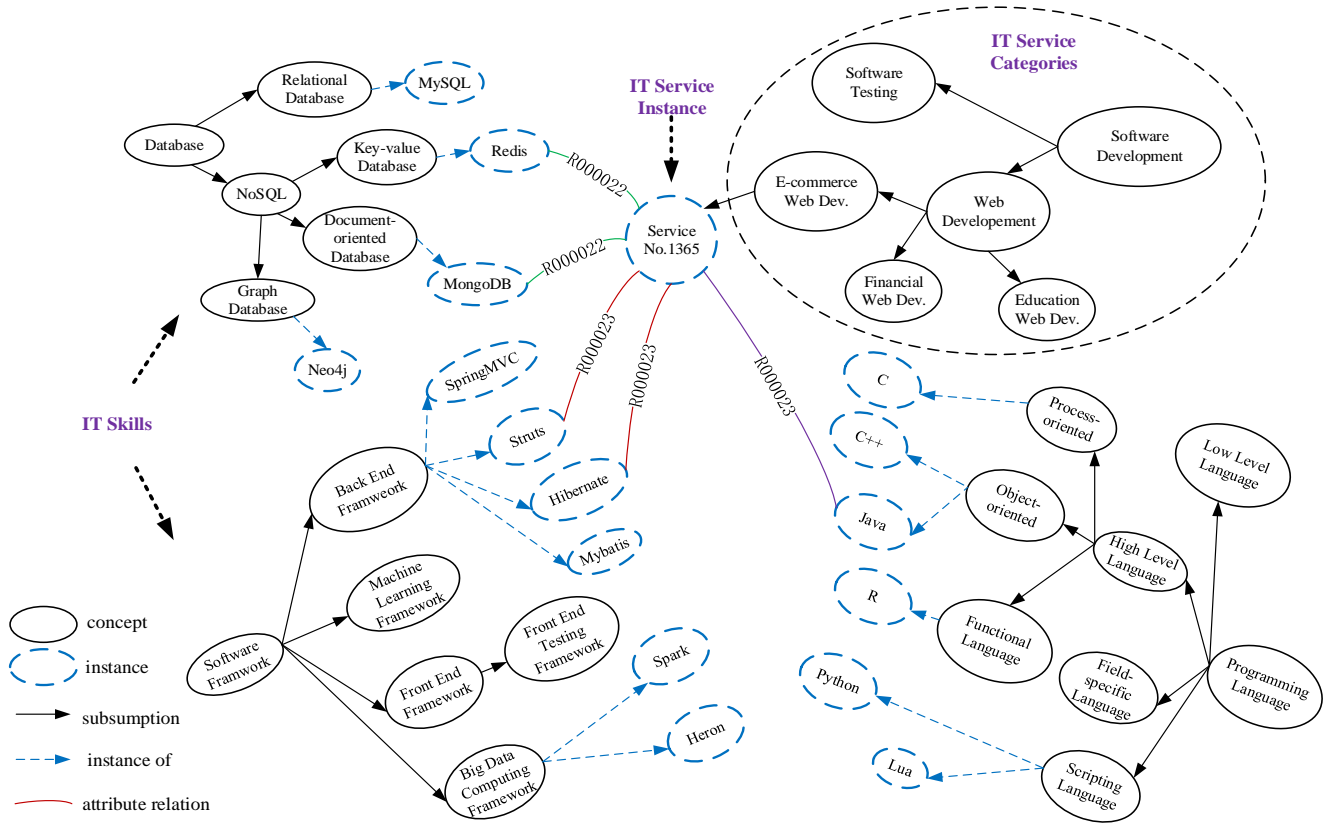


Figure 3. One fragment of IT Service Knowledge Graph

approach boosts query understanding, and alleviates the problem that service information lacks enough text resources.

- 2) *Service re-ranking*: After the preliminary search results are returned by the Elasticsearch engine, a learning-to-rank model is applied to re-rank the results and obtain the top-N IT services. Our approach designs several novel features, including degree of literal overlap, embedding vector similarity, query-service correlation, and service quality. Thus, it makes the more relevant service have the higher-ranking position.

Next explains the details of IT service knowledge base, term expansion and service re-ranking.

B. Building IT Service Knowledge Graph

A domain knowledge graph in IT crowdsourcing should include the information of IT services and the knowledge in IT service implementation. Therefore, we design ITServiceKG from three aspects: IT service categories, IT skills and IT service instances. In ITServiceKG, each entity (i.e. concept or instance) has following attributes: name, synonym, cooccurrence, and

description; and relations between entities includes subsumption, instance-of, and attribute relations. One fragment of ITServiceKG is shown in Figure 3.

1) *IT service categories*: IT service categories record the hierarchical structure of service categories. These data are provided by the real-world IT crowdsourcing platform—*JointForce*¹. It holds a three-layer structure. The first layer contains various types of software services, including those for software development, software testing, architecture design, DevOps deployment, logo design, etc. The second layer collects sub-types of specific IT services. Let "software development" be an example. Its child services include software services such as "web development", "App development", "embed system development", and so on. And the third layer places domain-oriented IT services.

2) *IT skills*: IT skills contain the technologies that IT services adopt, such as programming languages, frameworks and database. Every skill holds a multi-layer structure. Take "Database" as an example, it can be divided into "Relational Database" and "NoSQL Database", and MySQL is an instance of "Relational Database". IT skills data are collected mainly from StackOverflow tag synonym system², Wikipedia³ and StuQ skill map⁴. And the cooccurrence data is processed from search logs.

3) *IT service instances*: IT service instances are to-be-crowdsourced IT services in the platform. They are connected to IT service categories and IT skills. Taking an example in Figure

¹<https://www.jointforce.com>

²<http://stackoverflow.com/tags/synonyms>

³<https://www.wikipedia.org>

⁴<https://github.com/TeamStuQ/skill-map>

3, Service No.1365 is an instance of e-commerce website development, with Redis and MangoDB as databases, Java as programming language, and Struts and Hibernate as frameworks.

C. Term Expansion

Term expansion is a common technique in information retrieval (IR), but we reach two conclusions that implies space for improvement: a) in previous research, the query expansion resources are from synonyms and cooccurrence words. The relations of superordinates and hyponyms are neglected; b) most researchers only pay attention to the query expansion but neglect the document expansion. In IT service crowdsourcing platforms, there are not enough textual description for services, which impedes the performance of semantic search. Therefore, the expansion of both user queries and service information is indispensable, and we leverage ITServiceKG to expand them along the following four channels.

1) *Synonym and cooccurrence expansion.* The entities in the ITServiceKG have attributes "synonym" and "cooccurrence", and we use synonym and cooccurrence terms of the linked entity to expand the queries.

2) *Superordinates and hyponyms expansion.* The IT service categories in ITServiceKG have a hierarchical structure. For example, in Fig. 3, the entity "Software Framework" has a hyponym "Back End Framework", the latter has a hyponym "Hibernate", and "Hibernate" as a relation with Service No.1365. We expand both for queries and services information with superordinates and hyponyms of the linked entity in ITServiceKG. For example, if a user wants to search for IT services with back-end frameworks, but does not know any concrete name of it, we can expand this query by adding the hyponyms of back-end framework: "Struts", "Hibernate", etc. For another example, Service No.1365 has a relation with entity "Struts", we can expand the information of Service No.1365 by adding the superordinate of "Struts" - "Back End Framework".

3) *Using the descriptions of entities as pseudo query.* ITServiceKG contains rich textual descriptions about entities. Given an entity e , we use its name and description as pseudo query. This makes sense because the query is usually short and concise, only including two or three keywords usually. So we use the descriptions of entities as the pseudo query, without worrying that the expanded query is too long or introduces too many irrelevant information.

4) *Using the description of entities to expand service information.* It makes sense that the extra information of entities in ITServiceKG can help us understand the meaning of the service information. It is like a process of looking up words in the dictionary. When we read an article and find a new word, we may look up the word in the dictionary. The meaning of the new word will help us understand the article. The domain knowledge graph behaves in the similar way. The description of entities can be helpful for users who do not have the background knowledge about the entities.

D. Service Re-Ranking

After the term expansion, the expanded query and service information are obtained. We build the reverse index in the

Elastic search engine, and get the preliminary results. Then, we apply machine learning techniques to re-rank the list of the search results and make the more relevant result have the higher-ranking position. This rank model can be viewed as a supervised learning problem. The input is the query and all to-be-crowdsourced IT services, the output is the top-N IT services, each of which has a relevant score of 0-1.

We adopt Gradient Decision Tree (GBDT) as our learning-to-rank model. It is a point wise ranking model. We design four features for this rank model, shown as below:

1) *The degree of literal overlap.* If the query and the service's name and description have more common characters, they are more relevant.

$$d = \frac{S_q \cap S_t}{|S_q \cup S_t|},$$

where S_q denotes the character set of a query and S_t denotes the character set of a service name.

2) *Embedding vector similarity.* We embed the query and the service's name and description into vectors by TD-IDF. The cosine similarity of the vectors between the query and the service indicates the relevance.

$$sim = \cos(v_q, v_t),$$

where v_q denotes the query vector and v_t denotes the vector of the service.

3) *Query-service correlation.* In IT service crowdsourcing platform, we record user service clicking events on search results as $\langle \text{query}, \text{service}, \text{timestamp} \rangle$ triples in the log. More clicking times indicates higher relevance.

4) *Service quality.* User quality evaluation on crowdsourced services are also recorded in IT service crowdsourcing platform. People tend to search for services with good quality, so it makes sense that we give them higher relevance score.

$$quality = w_p \cdot p + w_s \cdot s + w_t \cdot t,$$

where t denotes the score of service on-time delivery, and s denotes the product quality score, t denotes the technical support score, and w is the corresponding weight.

We transfer the query-service pair to feature vectors and use them as the input of the GBDT model. Then a point wise ranking model is learned through model training on historical data, and re-ranks the current search results.

IV. EXPERIMENTS

We implemented our semantic search approach, and conducted evaluations to explore the following research questions:

(RQ1): What is the effectiveness of our approach compared with the baseline?

(RQ2): How much does each expansion channel contribute to IT service search?

TABLE I. THE OVERALL PERFORMANCE COMPARISON

Methods	P@10(%)	Δ	R@10(%)	Δ	P@15(%)	Δ	R@15(%)	Δ	MRR	Δ
TF-IDF	13.096	—	49.563	—	9.856	—	56.396	—	0.542	—
BM25	13.107	+0.084	51.872	+4.659	9.905	+0.497	57.197	+1.420	0.553	+2.030
term expansion	13.053	-0.328	52.369	+5.661	9.996	+1.420	61.263	+8.630	0.695	+28.229
term expansion + service re-ranking	13.298	+1.542	52.965	+6.864	10.236	+3.856	62.586	+10.976	0.731	+34.871

(RQ3): How much does each feature contribute to IT service re-ranking?

A. Experiment Setup

1) *Dataset*. Experimental data is offered by JointForce, the biggest IT crowdsourcing platform in China. There are 8753 services and 18025 search records. Each record in search logs contains a query (q), top k service list returned by search engine ($Sq@k$), and service list clicked by user (Iq). We split the dataset into training (80%) and testing (20%).

2) *Evaluation Metrics*. We use precision (P@ k), recall (R@ k) and MRR (Mean Reciprocal Rank) to measure the performance of our approach. For query sets Q , these metrics are defined as follows:

Precision (P@ k): It is the percentage of correctly discovered services in all discovered k services.

$$P@k = \frac{1}{|Q|} \sum_{q \in Q} \frac{|Sq@k \cap Iq|}{|Sq@k|}$$

Recall (R@ k): It is the percentage of correctly discovered services in all correct services.

$$R@k = \frac{1}{|Q|} \sum_{q \in Q} \frac{|Sq@k \cap Iq|}{|Iq|}$$

MRR: The reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer: 1 for first place, 1/2 for second place, 1/3 for third place and so on. The MRR is the average of the reciprocal ranks of results for a sample of the query sets Q :

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{rank_q},$$

where $rank_q$ refers to the rank position of the first relevant service for the query q .

B. RQ1. Overall Performance

Basic search approaches with BM25 and TD-IDF are selected for overall performance comparison. The experimental result is shown in the Table I. We can observe that the performance of BM25 is better than TD-IDF, therefore, we choose BM25 as the reference in the subsequent experiments. Moreover, we can observe that when applying term expansion, the recall is enhanced more prominently than the precision. It makes sense because the term expansion introduces more useful relevant terms and can find more correct services which cannot be retrieved before. After further applying ranking model, the precision is enhanced, because we make the more relevant services have higher ranking position. The performance of @15

is better than @10, which means @15 is a balanced metric for both precision and recall.

The experiment demonstrates that our approach (i.e. basic search with BM25 + term expansion + service re-ranking) outperforms the baselines. The precision is improved by 3.856%, the recall is improved by 9.912%, and the MRR is improved by 34.871%. This is because we introduce extra domain knowledge and retrieve some results that are neglected in the baseline approaches.

C. RQ2. Expansion Channel Contribution Analysis

During term expansion, our approach adopts four expansion channels: synonym and cooccurrence expansion (SCQ), superordinates and hyponyms expansion (SHQ), descriptions of entities as pseudo query (DQ), and descriptions of entities to expand service information (DS). In this experiment, we perform an analysis to evaluate each channel's contribution to the performance of IT service search.

We choose basic search approach with BM25 as the baseline, and add each expansion channel one by one. Recall@15 is chosen as the evaluation metric, because term expansion mainly enhances the recall.

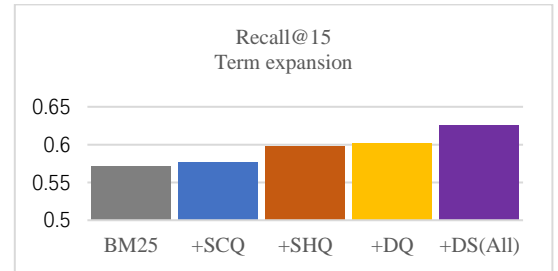


Figure 4. Expansion Channel Contribution Analysis

The impact of each expansion channel on the overall performance in this experiment is shown in Figure 4. We can observe that the performance is improved greatly when introducing "Superordinates and hyponyms expansion" and "Using description of entities to expand service information". We take following two examples to illustrate why superordinates and hyponyms can improve the performance.

The first query example is "ERP web service that uses Struts and Hibernate". Before term expansion, we find some related services, all of them relying on exact match. But Struts is not as popular as Spring now, so the services using Struts are rare. When we introduce the superordinates of entity "Struts"-

"Back End Framework" and add it to the query, we can find some services with back-end framework, such as using SpringMVC and Mybatis.

The second query example is "student management system that uses NoSQL". Before applying term expansion, we cannot find services with keyword "NoSQL", because "NoSQL" does not appear in the name or description of any service, since more detailed words are used to describe their techniques, such as "Redis", "MongoDB" rather than the general word "NoSQL". Therefore, we introduce the hyponyms of the entity "NoSQL", and then we can find services use Neo4j and MongoDB, which are both NoSQL databases.

We can also observe that when we use the description of entities to expand service information, the performance is improved greatly. The reason is that there are not enough available textual descriptions for services, and after we use description of entities to expand them, this problem is alleviated, and thus the performance becomes better.

D. RQ3. Ranking Feature Contribution Analysis

Our approach adopts four features for the learning-to-ranking model: degree of literal overlap (DLV), embedding vectors similarity (EVS), query-service correlation (QSC) and service quality (SQ). In this experiment, we perform an analysis to evaluate each feature's contribution to the performance of ranking model.

The baseline is the basic approach with term expansion. The metric is Recall@15. We add the features one by one and the impact of each feature on the performance is shown in Figure 5.

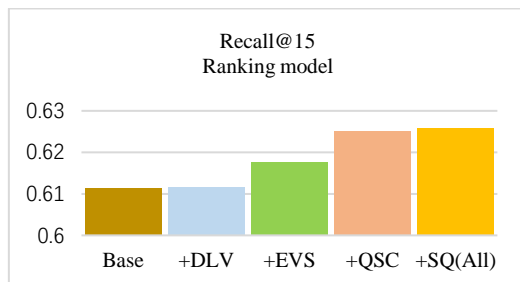


Figure 5. Ranking Feature Contribution Analysis

From Figure 5 we can see that all the features take effect. The embedding vectors similarity and the query-service correlation contribute most in the four features. The embedding vectors similarity mainly represents the similarity between the queries and the name of services. This implies the name of service plays an important role in the model. The query-service correlation implies how correlated a query and a service is in the history of previous crowdsourcing, so it can give advice to the search in the present.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a knowledge graph approach to enhancing semantic search for crowdsourced IT services. Compared with the traditional search approach, the MRR of our approach increases 34.871% and the Recall@15 increases

10.976%. The key to success comes from two aspects: (1) an IT service knowledge graph is built and utilized to expand both user queries and service information; (2) a learning-to-rank model with four features is designed to re-rank the preliminary search results.

As for future work, we will employ neural networks and learn latent representations of words, entities, and their relations in the knowledge base. These latent representations can be learnt in an unsupervised manner to be subsequently leveraged in a ranking model.

ACKNOWLEDGMENT

This research is supported by 973 Program in China (Grant No. 2015CB352203), National Nature Science Foundation of China (Grant No. 61472242 and 61572312), and Shanghai Municipal Commission of Economy and Informatization (No. 201701052). Thanks JointForce for providing the experimental data set.

REFERENCES

- [1] Y. Sun, A. Singla, D. Fox, and A. Krause. Building hierarchies of concepts via crowdsourcing. *arXiv preprint arXiv:1504.07302* (2015).
- [2] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment* 5, 11 (July 2012), 1483-1494.
- [3] Runtao Qiao, Shuhan Yan and Beijun Shen, A Reinforcement Learning Solution to Cold-Start Problem in Software Crowdsourcing Recommendations. In *International Conference on Progress in Informatics and Computing (PIC)* 2018.
- [4] Bhogal J, Macfarlane A, Smith P. A review of ontology based query expansion. *Information Processing & Management* 43, 4 (July 2007), 866-886.
- [5] Wei Xing, Peng F, Dumoulin B. Analyzing web text association to disambiguate abbreviation in queries. In *SIGIR* 2008. ACM, 751-752.
- [6] Derczynski L, Wang Jun, Gaizauskas R, et al. A Data Driven Approach to Query Expansion in Question Answering. *arXiv preprint arXiv:1203.5084* (2012).
- [7] B Mitra, N Craswell, Neural models for information retrieval. *arxiv preprint arXiv:1705.01509* (2017).
- [8] Chenyan Xiong. Explicit Semantic Ranking for Academic Search via Knowledge Graph Embedding. In *WWW* 2017. ACM, 1271-1279.
- [9] Laura Dietz, Chenyan Xiong, Edgar Meij. The First Workshop on Knowledge Graphs and Semantics for Text Retrieval and Analysis (KG4IR). In *SIGIR* 2017. ACM, 1427-1428.
- [10] Alexander Kotov, ChengXiang Zhai. Tapping into knowledge base for concept feedback leveraging conceptnet to improve search results for difficult queries. In *WSDM* 2012. ACM, 403-412.
- [11] Jeffrey Dalton, Laura Dietz, James Allan, Entity query feature expansion using knowledge base links. In *SIGIR* 2014. ACM, 365-374.
- [12] Chenyan Xiong, Jamie Callan. Esdrank: Connecting query and documents through external semi-structured data. In *CIKM* 2015. ACM, 951-960.
- [13] Xiangling Zhang, Yueguo Chen, Jun Chen, et al. Entity Set Expansion via Knowledge Graphs. In *SIGIR* 2017. ACM, 1101-1104.
- [14] Chenyan Xiong, Jamie Callan, Tie-Yan Liu. Word-Entity Duet Representations for Document Ranking. In *SIGIR* 2017. ACM, 763-772.
- [15] Wen Zhang, Bibek Paudel, Wei Zhang, Abraham Bernstein, Huajun Chen, Interaction Embeddings for Prediction and Explanation in Knowledge Graphs. In *WSDM* 2019. ACM, 96-104.

An Empirical Study on Research and Developmental Opportunities in Refactoring Practices

Shivani Jain

University School of Information, Communication and
Technology, GGS Indraprastha University
Sector 16 C, Dwarka
Delhi, India
shivani.1091@gmail.com

Anju Saha

University School of Information, Communication and
Technology, GGS Indraprastha University
Sector 16 C, Dwarka
Delhi, India
anju_kochhar@yahoo.com

Abstract—Maintaining large complex software is one of the major challenges faced by today's software industry. Refactoring is one way to do so. It is the process of changing internal structure of project code or software design without altering functionality. It improves software quality and reduces software entropy. This paper presents the preliminary results of an explanatory survey targeted at investigating refactoring practices by IT professionals. 221 participants helped to reveal important facts about refactoring risks, benefits, limitations of tools, and how a team manages consistency between different artefacts while practising refactoring. Findings reveal that refactoring tools are under-used as they have availability, usability and trust issues. An automated system is the need of the hour to manage change consistencies, visualizing code structures, detecting code, and design smells, and performing refactorings. This study will enable researchers and developers to understand their role in a better way as prevailing issues with current state-of-art are exposed and challenges are reported.

Keywords: *Refactoring; code smells; refactoring tools; survey; empirical study.*

I. INTRODUCTION

Code smells are design flaws, though are free of syntax errors, but can lead to future bugs and errors [1]. They are a violation of basic design principles and are also known as anti-patterns [2]. They contribute to financial debts and make software complex, hard to understand and maintain, and make changes difficult to embed. Thus, detection of such designs flaws and their correction is an absolute necessity [3]. Refactoring is one of the techniques to remove these anomalies. Refactoring is a well-organized practice for the reorganization of the present body of code, changing its internal structure without changing its external conduct. It is a sequence of little functionality preserving transformations. It is done while adding a feature, fixing a bug, and during code review [4].

Refactoring is mainly done to improve the internal structure and readability of the software. It increases flexibility, maintainability and reduces inter-modular couplings [5, 18]. It

can be done by a specialist or stakeholders like software designer, developer, tester or maintenance team. It can be done either manually or with the help of a tool. The scope can be system-wide or small scale, depending on the aim of applying refactorings. Refactoring is a time-consuming process that does not reflect immediate benefits like new features or bug fixes [6]. Incomplete and incorrect refactorings can lead to bugs [7] and it has been found that a high proportion of refactorings often led to an increase in the number of errors [8, 17].

The refactoring process [9] consists of following activities: (1) identify code smells; (2) determine which refactorings are best suitable for application; (3) make sure that applied refactoring preserves behaviour; (4) apply refactorings; (5) calculate the impact of refactorings on software quality features; (6) maintain uniformity between refactored program code and other software relics (test data, documents etc.).

Murphy-Hill [10] mentioned four different ways to collect research data for refactoring. They are:

- Mining the Commit Log – Look for mention of the word “refactor” in the commit logs of versioned repositories. Commit logs are updated when a programmer commits a change to the repository.
- Analyse Code Histories - Analyze an order of versions of the source code by manual comparison or by automating the comparison using a software tool.
- Observing Programmers - Observe developers in the field, working on software development and illustrate their refactoring behaviour. Such observation can be direct observation which comes under the category of a controlled experiment. Another is indirect observation which can be a survey or a project post-mortem.
- Logging Refactoring Tools - Some programming environments automatically record the programmer's activity in a log file. Such an environment is specialized to collect refactoring tool events.

DOI reference number: 10.18293/SEKE2019-038

In this study, the third method i.e. indirect observation has been implemented and data has been collected through a survey.

In this study, the following research questions have been addressed:

RQ1: How do programmers ensure that code has been refactored correctly and what are the measures taken to manage consistency between software artefacts?

RQ2: What are the reasons that prompt the refactoring process?

RQ3: What are the common refactoring practices followed by developers?

RQ4: Which are the most desirable features and barriers in the adoption of refactoring tools?

RQ5: What are the risks and benefits associated with refactoring?

To find answers to these burning questions, a survey was conducted and 221 software engineers participated in the study. Responses were collected online and analyzed quantitatively. The results were presented pictorially through graphs.

This study will make the following contributions:

- Study will assist researchers to identify research areas in the field of refactoring and to focus on issues to be solved in refactoring process. It will support the developers to develop the tools keeping in mind the shortcoming of the available tools. And will help IT professionals to understand the importance of refactoring and make it a part of their development process in various projects.
- Learning limitations of refactoring tools will guide researchers to focus on grey areas and what are prominent research areas, for example validation and verification of applied refactorings, maintaining change consistency in between artefacts, and development of better algorithms for detection of code smells etc. Developers can build easy to understand tools with better GUI and work on availability issues as well by creating awareness among development community.

II. RELATED WORK

G. H. Pinto and F. Kamei [11] did a qualitative and quantitative study to find out answers to following research questions: Which are the most desirable features in refactoring tools? What are the factors that prevent developers to adopt refactoring tools? Does interest in refactoring tools increase over the years? To uncover the number of issues regarding these tools, more than 1,400 messages – 324 questions and 1,115 answers to those questions were analyzed from more than 1,200 users. Major findings of this study are: refactorings tools are in demand for dynamic languages, databases and multi-language

refactorings. Users reported that a lack of trust and usability problems in tools still prevails. Interest in refactoring tools over the years has not increased as expected.

M. Kim, T. Zimmermann and N. Nagappan [12] conducted a survey at Microsoft, followed by a semi-structured interview and quantitative analysis of version history data of Windows 7 to reveal refactoring benefits and challenges. Survey finds that the refactoring definition in practice is not restricted to a standard definition of behaviour-preserving code alterations and developers observed that refactoring involves considerable cost and risks. The quantitative analysis of Windows 7 version history finds refactoring top 5% of modules led to a reduction in modular couplings and many complexity measures but increases the size more than the bottom 95%.

N. Singh and P. Singh [16] performed a comprehensive sentiments analysis on 3,171 GitHub comments during refactoring 60 open Java source projects by mining relevant commit messages. Research Question – “Does a refactoring task allocated during the implementation of a software feature following a strict deadline invoke positive or negative sentiments in the developer?” was answered. Tool RefTypeExtractor for automatically linking commit messages to their respective refactoring techniques was developed and dataset SentiRef, which stores the identified developer’s sentiments linked to each of 3,171 commit messages was created. The research concluded that in general software developers express more negative sentiments than positive sentiments while performing refactoring tasks which reveals the substandard state of the refactoring process.

Arcoverde, Roberta, Alessandro Garcia, and Eduardo Figueiredo [13] presented the results of a survey with the purpose of understanding the longevity of code smells in software projects. They concluded that (i) there is a probability of breaking APIs before refactorings by developers of widely-scoped reusable code; (ii) developers of standalone applications consider contract breaking changes easier to apply than developers of reusable assets; (iii) refactoring tools are more frequently used by developers that apply Test-Driven Development; (iv) refactoring tools are commonly used, and (v) reusable assets and standalone applications have different refactoring prioritization.

Our study intends to foster such previous investigations by revealing current challenges faced by developers and how they maintain consistency between different artefacts. We designed a questionnaire in order to understand the gap that subsists between the interpretation of refactoring practices by developers and researchers. The questionnaire was made available as an online survey and 221 software engineers filled it.

III. SURVEY SETTINGS

In order to understand the refactoring practices, a questionnaire was created and sent to 10 experts (well learned software engineers in the IT industry with more than 12 years

of experience in companies like Amazon, Adobe, Flipkart, TCS, Walmart etc.) and based on their feedback, the questionnaire was refined regarding the clarity and objectivity of the questions. It consisted of 14 multiple choice questions and 6 free-form questions. Few multiple choice questions had an option where participants could write answers of their choice as well. A glossary was included at the beginning explaining terms and acronyms used, for disambiguation. To collect information online survey was conducted and 221 engineers participated. Participants belonged to different companies and various designations. Majorly, they were developers (i.e. 90.5%) including requirement engineers, software designers, testers, researchers, full stack developers, and software architects etc. having a maximum of 20 years of experience, minimum of a year and an average of 3.2 years. The survey was divided into four sections and is described in Table 1.

The questions were formulated to identify how often and when refactorings are performed, what is the main purpose behind it and how consistency is managed between different artefacts while performing refactorings. Another section was focused on the most popular tools and what are the desirable features and barriers in the adoption of refactoring tools. Further, investigation on benefits, risks, and challenges regarding refactoring is explored. After the collection of responses, data was analyzed and categorized pictorially. Results that were revealed were both interesting and useful in understanding the roles and responsibility of researchers and developers in the field of refactoring. It revealed major challenges that still prevail and scope in the research area.

IV. RESULTS

The survey was made online and 221 IT professionals participated. 90.5% of them were developers from different companies and having experience in various languages like Java, Python, .NET etc. 23.4% of participants performed refactoring daily, 34.9% weekly and 21.1% daily. Software engineers from diverse and virtuous companies like Amazon, HCL, Infosys, Flipkart, Oracle, TCS, Expedia, Snapdeal, IBM, Paytm, ISRO, and ICAR etc. contributed to our findings. Participants had an average experience of 3.2 years with a maximum of 20 years and working in various nations like India, USA, Australia, Germany, and China.

The following section organizes the results in terms of the research questions.

RQ1: How do programmers ensure that code has been refactored correctly and what are the measures taken to manage consistency between software artefacts?

Results show that 78% of participants perform some kind of testing after refactoring code. Most of them prefer simple unit or functional testing but some of them prefer regression, integration, smoke, sanity or boundary value testing. Few of them make their peers to do code review. To maintain consistency, teams use version control platforms like Git, communicate to the team through a pre-defined channel or by

adding comments, maintain an excel file of changes. Test code and documents are changed manually after refactorings. Some of the remarks are as follows:

- *“Version control helps to keep a track of changes made, which once are completed successfully are documented.”*
- *“Rewrite/Update Unit Test, Update Documentation, Add relevant comments”*
- *“Code and test cases must go hand in hand. To ensure this I follow test based approach with unit test cases written before the code has to be refactored.”*
- *“We use XML notation for commenting code which reduces the need for separate documentation to a very large extent, code coverage tools for maintaining test cases.”*

Some of the good practices followed by engineers after performing refactorings are doing code reviews, compiling the code, running test cases, running bug detectors and modifying test cases according to the refactored code.

RQ2: What are the reasons that prompt the refactoring process?

Refactoring software is only beneficial when it is done with a purpose like reducing coupling between modules. Results revealed some of the reasons that prompt software engineers. 79% and 83% of professionals refactor when code gets hard to understand and maintain respectively. 50% agreed that slow performance and wide dependencies between modules are the main reason behind their refactoring actions. Logical mismatch, difficulty in debugging and testing, readability, re-usability and duplicity were also the main causes to initiate refactoring as depicted in Fig 1.

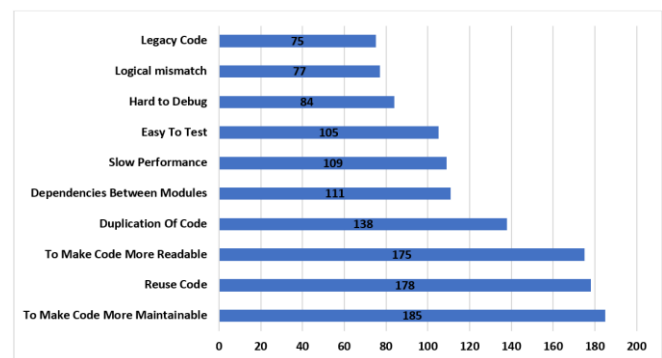


Figure 1. Reasons that prompt team to refactor code

Majority of the participants agreed that refactoring increases program flexibility, improves readability, reduces coupling, improves the internal structure of the code and makes it easier to add new features. Apart from these, honourable mentions were to reduce bugs, to increase consistency of an application, to reduce compiling time, to optimize and improve software performance, to enhance scalability and robustness of applications.

TABLE I. Summary of Survey Questions

Background	Which best describes your primary work area (developer, tester, manager etc.)? (open answer)
	How many years of work experience do you have in the software industry? (open answer)
	Name of Current Company and Country (open answer)
Refactoring Practices	How often do you perform refactoring (Daily, Weekly, Monthly, Yearly, Seldom, Never)? (multiple choice)
	Which keywords do you use or have you seen being used to mark refactoring activities in change commit messages? (multiple choice)
	How do you ensure that you have refactored program correctly? (open answer)
	How do you manage consistency between different software artefacts (e.g. documents, code and test cases) during refactoring? (open answer)
	What is the purpose of your refactorings? (multiple choice with the open answer)
	Which of these reasons prompts you to initiate the refactoring process? (multiple choice with the open answer)
	Select following options for refactorings [multiple choice: (a) manually and with a tool (b) manually, (c) using automated tools, (d) know this but don't use it, (e) don't know this refactoring.]
	• Rename, Extract Method, Encapsulate Field, Extract Interface, Remove Parameters, ... [From Fowler's catalogue]
	How do you strongly agree, agree, neither agree or disagree, disagree, strongly disagree with each of the following statements?
	• I perform refactorings with other functional changes. • Refactorings I want to perform are different from what supported by tools. • Tools do not support higher level refactorings. • How do you validate code refactorings?
Refactoring Tools	Few statements are shown in this table for presentation purposes.
	What tools do you use during refactoring? (open answer)
	How do you perform most of your refactorings? (multiple choice with the open answer)
	What are the barriers to adoption of refactoring tools? (multiple choice with the open answer)
Risks and Benefits	What are the features in refactoring tools you would like to have? (multiple choice with the open answer)
	How do you strongly agree, agree, neither agree or disagree, disagree, strongly disagree with each of the following statements?
	• Refactorings advance code readability • Refactorings introduce subtle errors • Refactorings disrupt other programmer's code • Refactorings advance performance • Refactorings make it debugging easy.
	What are the challenges associated with performing refactorings? (open answer)
	Based on your own experience, what are the risks involved in refactoring? (multiple choice with the open answer)
	What benefits have you observed from refactoring? (multiple choice with the open answer)
	Only some of the questions are mentioned for representation purpose.

RQ3: What are the common refactoring practices followed by developers?

Great proportion strongly agreed on the following practices:

- Refactorings are carried out in batches and changes in associated test cases and documents are reflected.
- Refactorings are done with other types of changes which modifies program behaviour externally. Pure refactorings are hardly done. This observation is consistent with R. Johnson's study [14].
- Refactorings that are done manually differ from what tools offer.
- Refactorings that are applied are higher level changes which are not supported by tools. This informs about the need for tools for higher level refactorings, for example dealing with generalization refactorings.
- Majority of the refactorings (60.6%) are done manually. This practice proves the urgent need for good quality, available and easy to use tools.
- Renaming, Extract Method and Remove Parameters are the most common refactorings performed manually or with the help of a tool. The same observation was made by M. Kim [15].

RQ4: Which are the most desirable features and barriers in the adoption of refactoring tools?

Participants listed a wide variety of tools that are used to perform common refactorings. Most commonly used tools are Jenkins, ReSharper, CodeRush, JS Refactor, Visual Assist X, TSLint, DPack, JetBrains etc. Refactorings like renaming and move method are simply done in IDEs like Eclipse, IntelliJ, Visual Studio etc.

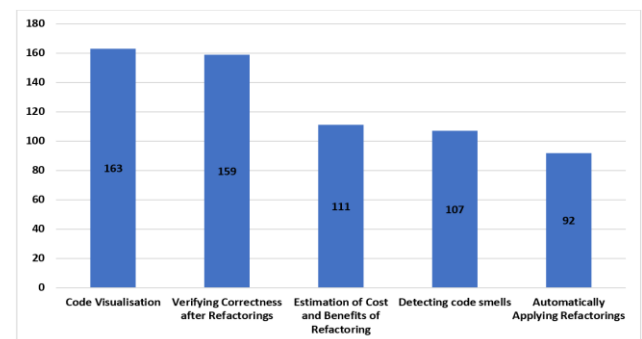


Figure 2. Most desirable features of refactoring tools

Most desirable features, developers want in refactoring tools are code smell detection feature, 74% want code visualization applicability. 72% of professionals would like to verify correctness feature after they are done performing refactorings, estimation of cost and benefits of refactoring are another requirement that participants mentioned. Automatically applying refactorings was only suggested by 42%. Fig 2 represents data in graphical form.

65% stated barriers that cease developers in adopting tools are less or no knowledge about the availability of tools. Around 29% participants mentioned difficult to understand/learn tools and unknown or not able to understand the debugging process is their reason that prevents them to use refactoring tools. Tools are not trustworthy and have bad GUI. Fig 3 represents same. Other mentions were:

- *“Company support”*
- *“Languages like Python and CSS have limited support for refactoring tools.”*
- *“They sometimes don't understand that “why I do, what I do” such as if I declare something in multiple lines, I mean it to be so, but formatting online size makes it hard to understand.”*

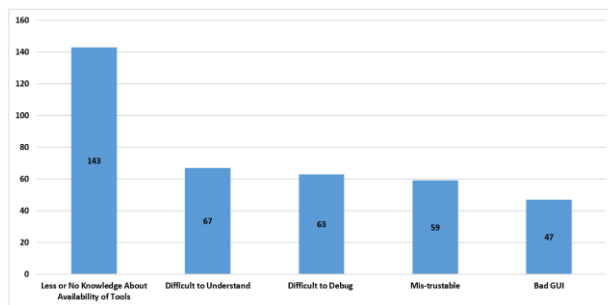


Figure 3. Barriers in the adoption of refactoring tools

RQ5: What are the risks and benefits associated with refactoring?

Refactoring risks are quantitatively presented through the graph in Fig 4. Key causes are identifying code smells. It is one of the major research areas in the field of refactoring. Many automatic, semi-automatic, and metric-based code smell detection techniques have and are being developed. Managing consistency between artefacts is a long and regressive process. Managing time is challenging as developers avoid refactoring code before major releases. Refactorings might introduce bugs or break existing code. Preserving behaviour, understanding legacy code, and convincing management team are considerable challenges.

Refactoring though is time-consuming procedure but it definitely yields promising benefits such as improved maintainability and readability which was further supported by more than 80% of the participants in this empirical study. More than 60% acknowledged improved performance, reduction in code size as well as duplicate code are perks of

refactoring code. Other advantages are the reduction in release time and bugs, software becomes easy to test and add the new features. Fig 5 represents the benefits of refactoring pictorially through a bar graph.

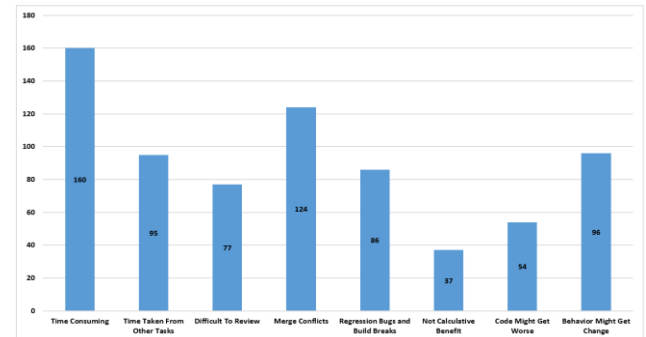


Figure 4. Risks involved in the refactoring process

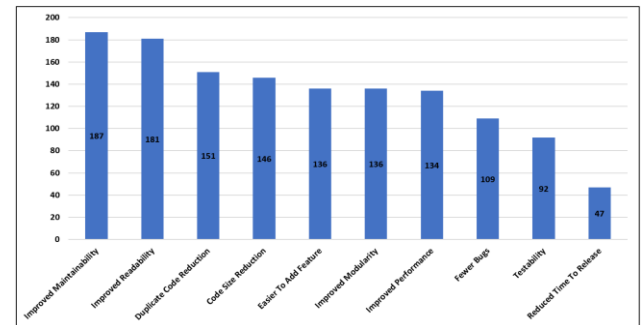


Figure 5. Benefits of performing refactorings

V. THREATS TO VALIDITY

Some points to be considered are:

Participants of survey conducted were IT professionals not refactoring specialists and with the assumption that people who filled the survey know what “Refactor” means. Survey made no inquiry about the type of projects (e.g. Web applications, Embedded systems, Information systems, etc.) participants had experience in. A number of participants were low to generalize the results and few of them had the experience below 5 years. Most of the questions were closed-ended which might lead to biasing.

VI. CONCLUSIONS AND FUTURE WORK

Large scale survey was conducted and a wide range of IT professionals was engaged. The main purpose of the survey was to understand the trends followed by developers and what are the research opportunities in the field of refactorings and developmental challenges. The study also answers the questions like what sort of tools should be built to better or automate the whole process and limitations of currently available tools. Survey responses were collected and analyzed to conclude the following results:

The refactoring process needs automated tools to maintain changes between different artefacts of software.

Some companies use version control systems but a system that updates the design, test cases, documentation automatically after refactoring is in demand. Software companies can invest their efforts in this sector.

Refactoring without purpose will not yield any benefits. Some of the factors that prompt refactorings are - hard to understand and maintain program code, wide inter-modular dependencies, difficulty in debugging and testing, readability issues etc. Software development teams should devote their time in refactoring process to overcome such issues.

Refactoring increases program flexibility, reduces coupling, improves the internal structure of the code and makes it easier to add new features. These points were supported by the majority of participants. Benefits associated with the refactoring process are improved maintainability, performance, modularity and readability, reduction in code size, duplicate code, release time and bugs, and easy to test. They are the motivations for teams to invest more time and effort in the refactoring process.

Common practices in refactoring are that they are carried out in batches, done with other types of changes that modify external program behaviour. Majority of the refactorings (60.6%) are done manually. Renaming, Extract Method, and Remove Parameters are the most common refactoring performed manually or with the help of a tool.

The reason that prevents software engineers to adopt refactoring tools is - less or no knowledge about the availability of tools, difficult to understand/learn, unknown or not able to understand debugging process, trust issues, and bad GUI. So, developers and researchers should investigate the reasons behind such an inappropriate condition of refactoring tools and try diminishing them. Tools for many dynamic languages are still unavailable. Code structure visualization, code smell detection, cost and effort estimation tools are coveted. Tools to validate and verify refactorings need to be developed.

Most desirable features, developers want in refactoring tools are automatic code smell detection, code visualization aspect, verifying correctness after refactoring edits, estimation of cost, efforts and benefits of refactoring, automatically applying refactorings. This provides research areas that should be explored by researchers and developers working in the field of refactoring.

Refactoring challenges faced by professionals are identifying code smells, managing consistency between artefacts, time management, excessive couplings between modules, refactorings might introduce bugs or break existing code, behaviour preservation, understanding legacy code, and convincing management team. Researchers can work in these domains to ease up the refactoring process. Companies and teams should realize the importance of refactoring code

and refactoring activities needs to be further encouraged to reduce maintenance time, money, and effort.

For future work, personal interview with professionals can be conducted on a large scale and further conclusions can be made by observing refactorings patterns and behaviour of IT teams in the field.

REFERENCES

- [1] V. Maggio and M. Faella, "Improving the Design of Existing code," Slides, 2011.
- [2] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Trans. Softw. Eng.*, vol. 30, no. 2, pp. 126–139, 2004.
- [3] M. Tufano et al., "When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away)," *IEEE Trans. Softw. Eng.*, vol. 43, no. 11, pp. 1063–1088, 2017.
- [4] M. Fowler, "Refactoring," p. 13472, 2002.
- [5] W. Opdyke, "Refactoring Object-Oriented Frameworks," PhD thesis, University of Illinois at Urbana-Champaign, 1992.
- [6] W. Opdyke, "Refactoring, reuse & reality," Lucent Technologies/Bell Labs, 1999.
- [7] C. Gørg and P. Weißgerber, "Error detection by refactoring reconstruction," in *Proc. Int. Workshop Mining Software Repositories*, 2005, pp. 1–5.
- [8] P. Weißgerber and S. Diehl, "Are refactorings less error-prone than other changes?," *Proc. ACM Int. Workshop Mining Software Repositories*, 2006, pp. 112–118.
- [9] A. V. D. Tom Mens, "Refactoring: Emerging Trends and Open Problems," *IEEE Int. Conf. Software Maintenance, ICSM*, pp. 521–522, 2003.
- [10] E. Murphy-Hill, Danny Dig, Chris Parnin, "Gathering refactoring data: a comparison of four methods," *Proc. 2nd Work. Refactoring Tools WRT 08 conjunction with Conf. Object Oriented Program. Syst. Lang. Appl. OOPSLA 2008*, 2008.
- [11] G. H. Pinto and F. Kamei, "What programmers say about refactoring tools?," *Proc. 2013 ACM Work. Work. refactoring tools - WRT '13*, pp. 33–36, 2013.
- [12] M. Kim, T. Zimmermann, and N. Nagappan, "An Empirical Study of Refactoring Challenges and Benefits at Microsoft," *IEEE Trans. Softw. Eng.*, vol. 40, no. 7, pp. 633–649, 2014.
- [13] Arcoverde, Roberta, Alessandro Garcia, and Eduardo Figueiredo. "Understanding the longevity of code smells preliminary results of an explanatory survey." *Proceedings of the 4th Workshop on Refactoring Tools*. ACM, 2011.
- [14] R. Johnson, "Beyond behavior preservation," *Microsoft Faculty Summit 2011, Invited Talk*, Jul. 2011.
- [15] Kim, Miryung, Thomas Zimmermann, and Nachiappan Nagappan. "A field study of refactoring challenges and benefits," *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012.
- [16] Singh, Navdeep, and Paramvir Singh, "How Do Code Refactoring Activities Impact Software Developers' Sentiments?-An Empirical Investigation Into GitHub Commits," *24th Asia-Pacific Software Engineering Conference*, pp. 648–653 IEEE, 2017.
- [17] Alshayeb, Mohammad, "Empirical investigation of refactoring effect on software quality," *Information and software technology* 51.9 (2009), 1319–1326.
- [18] Szőke, Gábor, Gábor Antal, Csaba Nagy, Rudolf Ferenc, and Tibor Gyimóthy, "Empirical study on refactoring large-scale industrial systems and its effects on maintainability," *Journal of Systems and Software* 129 (2017): 107–126.

An Empirical Study on Optimal Solutions Selection Strategies for Effort-Aware Just-in-Time Software Defect Prediction

Xingguang Yang^{*†}, Huiqun Yu^{*‡✉}, Guisheng Fan^{*✉}, Kang Yang^{*}

^{*}Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China

[†]Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai 201112, China

[‡]Shanghai Engineering Research Center of Smart Energy, Shanghai, China

Abstract—Just-in-time software defect prediction (JIT-SDP) is an active topic in the field of software engineering, and many methods have been proposed to solve this problem. State-of-the-art method MULTI applies multi-objective optimization algorithm to the effort-aware JIT-SDP problem, and obtains good average performance. Although the average performance of the MULTI method is high, there are many optimal solutions with poor performance. If an optimal solution is randomly selected, a poor prediction model may be obtained. In order to further improve the performance of the MULTI method, we propose three optimal solutions selection strategies: benefit priority (BP), cost priority (CP), and a compromise between cost and benefit (CCB). In order to compare and validate the effectiveness of the strategies, we conduct a large-scale empirical study on data sets of six open source projects. The experimental results show that, compared with the average performance of MULTI, the optimal solutions selection strategy based on BP has a significant improvement in ACC and P_{opt} indicators. Therefore, we recommend using the BP-based optimal solutions selection strategy to improve the performance of MULTI when using the MULTI method to solve the effort-aware JIT-SDP problem.

Index Terms—software defect prediction, empirical software engineering, multi-objective optimization algorithm, just-in-time, effort-aware

I. INTRODUCTION

Software defect prediction [1] [2] is an active research topic in the domain of software engineering. As software scales increase, it becomes very difficult to release a high-quality software system. In the case of limited software testing resources, it is critical for the enterprise to find and fix defects in the software as early as possible.

Software defect prediction is an effective method. By using defect prediction models, developers can accurately estimate whether a program module is likely to have defects, thereby allocating more test resources to modules that are more likely to have defects, which help to improve the quality of the software [3].

Traditional software defect predictions have shortcomings in practical applications [4]. First, because the prediction

granularity is coarse (i.e., class, file, or package), it is time consuming to locate risky code regions. Second, a module is usually completed by many developers, so it is difficult to find the suitable developer to code checking for the defect-prone module. Finally, because the defect prediction is made in the later stages of software development, it is difficult for developers to come up with thoughts for software development.

Just-in-time software defect prediction (JIT-SDP) can well overcome the above deficiencies. JIT-SDP is made at change-level rather than module-level. Once the developer submits the modified code, the defect prediction will be executed. Developers can quickly know whether the change is defect-inducing or not.

Nowadays, many supervised and unsupervised methods have been proposed to solve the JIT-SDP problem. Yang et al. [5] compared the performance of supervised learning methods and unsupervised learning methods in the context of effort-aware JIT-SDP through a large-scale empirical study. The experimental results show that some unsupervised learning methods are even better than supervised learning methods, which contradicts people's experience, because supervised learning methods can learn knowledge from data sets and should have better performance than unsupervised learning methods.

Therefore, in order to improve the performance of supervised learning methods, Chen et al. [6] proposed a new method called MULTI, which formalized the effort-aware JIT-SDP problem into a multi-objective optimization problem. The optimal solution set is obtained by using the classical multi-objective optimization algorithm (MOA) NSGA-II. The experimental results show that the average performance of the MULTI method is better than the state-of-the-art supervised learning methods and unsupervised learning methods. However, Chen et al. [6] used the median of the performance of the optimal solution set to represent the average performance of the MULTI method. In the actual use of the MULTI method, if an optimal solution is randomly selected, a poor performance prediction model may be obtained. Therefore, in order to improve the stability and performance of the MULTI

Corresponding Authors: Huiqun Yu (yhq@ecust.edu.cn), Guisheng Fan (gsfan@ecust.edu.cn) DOI reference number: 10.18293/SEKE2019-174

method, we design three optimal solutions selection strategies: benefit priority (BP), cost priority (CP), and a compromise between cost and benefit (CCB). Empirical results show that the optimal solutions selection strategy based on BP can significantly improve the performance of MULTI. Therefore, we recommend using the BP strategy to obtain a higher performance prediction model when using the MULTI method to solve the effort-aware JIT-SDP problem.

The contributions of this paper are summarized as follows:

- In view of the shortcomings of the MULTI method, we propose three optimal solutions selection strategies BP, CP, and CCB, aiming to improve the effort-aware prediction performance of the MULTI method.
- Through large-scale empirical research, we compare the performance of three optimal solutions selection strategies and the average performance of MULTI on the data sets of six open source projects. Experimental results demonstrate that BP-based optimal solutions selection strategy can significantly improve the performance of the MULTI method in ACC and P_{opt} indicators.

The rest of this article is as follows. Section II introduces related work of software defect prediction. Section III introduces the principle of MULTI and our proposed optimal solutions selection strategies. Case study is introduced in section IV. Experimental results and discussion are presented in section V. Section VI describes the threats to validity. The conclusion and future work are presented in section VII.

II. RELATED WORK

A. Just-in-time Software Defect Prediction

Mockus and Weiss [7] firstly applied JIT-SDP to 5ESS updates by designing a series of change metrics. Experimental results show that JIT-SDP can be effectively used in many commercial software projects. Kamei et al. [4] conducted a large-scale empirical study of 11 projects, including 6 open source projects and 5 commercial projects, and they shared data sets of 6 open source projects. Empirical results reveal that the model based on logistic regression can achieve 68% accuracy and an average recall rate of 64%.

Subsequently, many methods were proposed to improve the performance of the JIT-SDP models. Yang et al. [8] proposed a novel method called TLEL, which leverages decision tree and ensemble learning. Experimental results indicate that their method can significantly improve the performance of JIT-SDP. McIntosh et al. [3] explored the impact of data validity on the performance of the JIT-SDP models. Empirical results demonstrate that in order to ensure the performance of the prediction models, the defect data sets used should be within the last three months.

B. Effort-aware Defect Prediction

When building a defect prediction model, in addition to considering the precision, recall, accuracy, etc., it is also necessary to consider the effort required to code checking for defect-prone models. Kamei et al. [4] applied effort-aware defect prediction to JIT-SDT. They proposed a new method EALR,

which can identify 35% buggy changes while 20% effort is used. Yang et al. [5] compared unsupervised models with supervised models for effort-aware JIT-SDP. Experimental results show that some unsupervised models outperform state-of-the-art supervised models for effort-aware JIT-SDP. Later, Fu and Menzies [9] repeated the experiment of Yang et al. [5] and proved that although supervised models are better than unsupervised models in project-by-project-based verification, supervised models are not superior to unsupervised models in general.

III. OPTIMAL SOLUTIONS SELECTION STRATEGIES

In order to improve the prediction performance of supervised methods in effort-aware JIT-SDP, Chen et al. [6] proposed a novel method MULTI, which applies multi-objective optimization algorithms (MOAs) to JIT-SDP. Their experimental results show that the average performance of MULTI is significantly better than the 43 state-of-the-art methods including 31 supervised methods and 12 unsupervised methods.

However, we find that the Pareto optimal set derived from MULTI has many optimal solutions with poor performance, which affects the performance of the MULTI method. Therefore, we propose three optimal solutions selection strategies designed to improve the performance of the MULTI method.

A. MULTI

Inspired by search based software engineering (SBSE) [10], Chen et al. [6] first formalized the effort-aware JIT-SDP problem into a multi-objective optimization problem. SBSE aims to solve complex problems with large-scale search space in software engineering by using search technology.

1) *The Design of Objectives*: MULTI uses logistic regression for defect prediction, which is widely used in previous studies [4] [11]. Assuming that a change $c = \langle m_1, m_2, \dots, m_n \rangle$ has n metrics, the prediction process of the logistic regression models can be denoted by the formula 1, where $w = \langle w_0, \dots, w_n \rangle$ is the coefficient vector of the logistic regression model. The output value of $y(c)$ represents the probability that a change c is buggy.

$$y(c) = \frac{1}{1 + e^{-(w_0 + w_1 m_1 + \dots + w_n m_n)}} \quad (1)$$

Since JIT-SDP is a bi-classification problem, we convert the output value of $y(c)$. The rule is as in formula 2. When the y value is greater than 0.5, the change c is classified as buggy, otherwise it is classified as clean.

$$Y(c) = \begin{cases} 1 & \text{if } y(c) > 0.5 \\ 0 & \text{if } y(c) \leq 0.5 \end{cases} \quad (2)$$

For the effort-aware JIT-SDP problem, MULTI mainly considers two optimization objectives based on cost-benefit analysis [6]. The first objective is designed from the perspective of benefit and to identify as many buggy changes as possible. For a set of changes C , the benefit of a model can be calculated by formula 3, which represents the number of buggy changes identified by the model.

$$benefit(C) = \sum_{c_i \in C} Y(c_i) \times buggy(c_i) \quad (3)$$

The return value of the function $buggy(c_i)$ indicates whether the change c_i is defect-inducing. When the change is buggy, the return value is 1 otherwise it returns 0.

The second objective is designed from the cost of the model and is to minimize the effort used for code checking. Once a change is predicted to be buggy by the defect prediction model, the software quality assurance (SQA) team will invest a lot of effort in code checking and test case design. For a set of changes C , the cost value can be calculated by formula 4. Here the function $SQA(c_i)$ represents the effort required to code checking for the change c_i . According to the suggestion of Kamei et al. [4], the value of $SQA(c_i)$ can be set to the lines of code(LOC) modified by the change c_i .

$$cost(C) = \sum_{c_i \in C} Y(c_i) \times SQA(c_i) \quad (4)$$

2) *The Generation of Optimal Solutions:* Obviously, the above two objectives are usually conflicting. If a model wants to identify more buggy changes, it will lead to more effort. Conversely, if the model wants to reduce the effort for code checking, it will miss many buggy changes. MULTI is designed based on NSGA-II [12], which is one of classical MOAs. Before introducing the coding scheme of chromosomes of MULTI, we give some definitions of MOAs.

- **Pareto dominance.** Suppose w_i and w_j are two feasible solutions of the JIT-SDP problem. If and only if $benefit(w_i) > benefit(w_j)$ and $cost(w_i) \leq cost(w_j)$ or $benefit(w_i) \geq benefit(w_j)$ and $cost(w_i) < cost(w_j)$, w_i is Pareto dominance on w_j .
- **Pareto optimal solution.** For a feasible solution w , w is a Pareto optimal solution if and only if there is no feasible solution w^* which is dominance on w . Pareto optimal set is composed by all the Pareto optimal solutions.

The process of MULTI can be summarized into the following four steps.

- 1) **Population initialization.** For the effort-aware JIT-SDP problem, a chromosome can be encoded as a coefficient vector, denoting the coefficients of a logistic regression model. During the population initialization process, N chromosomes are randomly generated, and the values of the elements of each chromosome are randomly generated.
- 2) **Evolution.** After population initialization, classical evolutionary operations are performed to generate new chromosomes. General evolutionary operators include crossover operator, mutation operator, etc.
- 3) **Selection.** Choose the optimal chromosomes from the parent and offspring populations. The selection operator of NSGA-II is based on non-dominated sorting algorithm and the concept of crowding distance [12]. Repeat steps 2 and 3.

- 4) **Termination.** Once the evolutionary process satisfies the termination condition, the iteration process terminates and returns to the final optimal chromosomes.

B. Optimal Solutions Selection Strategies

MULTI designs two optimization objectives and generates a set of optimal solutions based on NSGA-II. In previous studies, Chen et al. [6] use MULTI-B to indicate the best performance of MULTI, and use MULTI-M to indicate the average performance of MULTI. Although the average performance of the MULTI method is good in the context of effort-aware JIT-SDP, randomly selecting an optimal solution from the optimal solution set may result in poor prediction performance.

Fig. 1 shows the values of ACC indicator generated from a run results on six subject systems based on 10 times 10-fold cross-validation. It can be seen from the Fig. 1 that although the median of indicator ACC in optimal solution set can obtain a high performance, there are a large number of poor performance solutions in the optimal solution set. Therefore, randomly selecting an optimal solution may result in obtaining a prediction model with poor performance. Therefore, it is necessary to design a suitable optimal solutions selection strategy to improve the stability of MULTI.

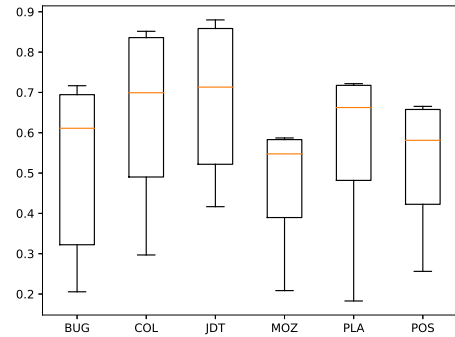


Fig. 1. values of ACC indicator for six subject systems

Since MULTI considers two optimization objectives, we design three optimal solutions selection strategies, as follows:

- **Benefit priority (BP).** BP strategy returns the optimal solution with maximum benefits in the optimal solution set.
- **Cost priority (CP).** CP strategy returns the optimal solution with minimal cost in the optimal solution set.
- **A compromise between cost and benefit (CCB).** CCB strategy considers cost and benefit simultaneously, and returns the optimal solution with middle cost or benefits in the optimal solution set.

To better describe the three strategies, we use the pseudo code to further describe them, as shown in Algorithm 1.

IV. CASE STUDY

Our case study aims to solve following research question.

Algorithm 1: Three optimal solutions selection strategies**Input:** training set: $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$;**Output:** optimal solutions: $solution_BP, solution_CP, solution_CCB$

```

1 begin
2   // Generate optimal solution based on MULTI method
3    $solutions = MULTI(D)$ 
4   // Sort optimal solutions in ascending order based on
   benefit values
5    $solutions = order\_by\_benefit(solutions)$ 
6   // Return solutions according to three strategies
7    $solution\_BP = solutions[solutions.length - 1]$ 
8    $solution\_CP = solutions[0]$ 
9    $solution\_CCB =$ 
    $solutions[(solutions.length - 1)/2]$ 
10 end

```

- Which optimal solutions selection strategy is appropriate for solving the effort-aware JIT-SDP problem?

The experimental hardware environment is *Intel(R) Core(TM) i7-7700 CPU@ 3.60GHz; RAM 8.00GB*. The experimental code is written in python.

This section introduces data sets, performance indicators, data analysis method, and experimental design.

A. Data Sets

The experiment considers the data sets of six open source projects shared by Kamei et al. [4], which have been widely used in previous studies [5] [6]. These six data sets include Bugzilla (BUG), Columba (COL), Eclipse JDT (JDT), Eclipse Platform (PLA), Mozilla (MOZ), and PostgreSQL (POS), and come from different domains with different scales. The basic information is shown in the Table I.

In order to better solve the JIT-SDP problem, 14 change metrics are designed for these data sets, as shown in Table II. These metrics can be divided into five dimensions: diffusion, size, purpose, history, and experience. More details can be found in reference [4].

TABLE I
THE BASIC INFORMATION OF DATA SETS

Project	Period	#defective changes	#changes	%defect rate
BUG	1998/08/26~2006/12/16	1696	4620	36.71%
COL	2002/11/25~2006/07/27	1361	4455	30.55%
JDT	2001/05/02~2007/12/31	5089	35386	14.38%
MOZ	2000/01/01~2006/12/17	5149	98275	5.24%
PLA	2001/05/02~2007/12/31	9452	64250	14.71%
POS	1996/07/09~2010/05/15	5119	20431	25.06%

B. Performance Indicators

Effort-aware JIT-SDP primarily considers effort for code inspection of defect-prone changes. According to the suggestion of Kamei et al. [4], the effort used to check changes

TABLE II
THE DESCRIPTION OF METRICS

Dimension	Metric	Description
Diffusion	NS	Number of modified subsystems
	ND	Number of modified directories
	NF	Number of modified files
	Entropy	Distribution of modified code across each file
Size	LA	Lines of code added
	LD	Lines of code deleted
	LT	Lines of code in a file before the change
Purpose	FIX	Whether or not the change is a defect fix
	NDEV	Number of developers that changed the files
History	AGE	Average time interval between the last and the current change
	NUC	Number of unique last changes to the files
	EXP	Developer experience
Experience	REXP	Recent developer experience
	SEXP	Developer experience on a subsystem

can be obtained by calculating their code churn (i.e., the total number of lines added and deleted by the change). Similar to previous studies [4] [5] [6], our experiment use ACC and P_{opt} to evaluate the effort-aware prediction performance for prediction models. ACC indicates the recall of bug changes when using 20% of effort. P_{opt} is the normalized version of effort-aware performance indicator proposed by Mende and Koschke [13]. Specific information on these two performance indicators can be found in the literature [4] [5].

C. Data Analysis Method

The experiment uses 10 times 10-fold cross-validation technique to evaluate the performance of prediction models. 10 times 10-fold cross-validation technique is performed within the same project. First, the data set of one project is randomly divided into 10 sets of the same scale, nine of which are used to train the model and produce the optimal solution set, and the other one is to test the performance of the optimal solution set. This step will be repeated 10 times. In our case study, the optimal solutions selection strategies only select one solution from optimal solution set. Therefore, 10 times 10-fold cross-validation can ultimately return 100 solutions for each strategy.

In order to test the degree of performance difference between different optimal solutions selection strategies, the experiment uses Wilcoxon signed-rank test and Cliff's δ to further analyze the experimental results. Wilcoxon signed-rank test is a commonly used non-parametric statistical hypothesis test method. In particular, we use corresponding p-values to exam whether two optimal solutions selection strategies have significant difference at the significance level of 0.05. Meanwhile, we use Cliff's δ to determine the magnitude of difference in practical application [14]. Traditionally, the magnitude of the difference is considered trivial ($|\delta| < 0.147$), small ($0.147 \leq |\delta| < 0.33$), moderate ($0.33 \leq |\delta| < 0.474$), or large ($|\delta| \geq 0.474$).

D. Experimental Design

- **Data preprocessing.** In order to obtain a better prediction model, according to the suggestion of Kamei et al. [4], we preprocess the experimental data sets as following steps.

- 1) ND and REXP metrics are excluded since NF and ND, REXP and EXP are highly correlated. LA and LD metrics are normalized by dividing by LT metric since LA and LD are highly correlated. LT and NUC metrics are also normalized by dividing NF since LT and NUC have highly correlation with NF.
- 2) Each metric (except FIX) is executed with logarithmic transformation, since these metrics are highly skewed.
- 3) Due to class imbalance in the defect data sets, we perform random undersampling and keep the number of clean changes same as the number of buggy changes by deleting clean changes randomly.

Our optimal solutions selection strategies are based on the MULTI method, which uses NSGA-II, so some parameters need to be set.

• **Parameter settings.**

- 1) Population size: 200.
- 2) The range of coefficient vector: [-10000, 10000].
- 3) The interval of population initialization: [-10, 10].
- 4) Crossover operation: simulated binary crossover.
- 5) Mutation operation: polynomial mutation.
- 6) The number of generations: 800.

V. EXPERIMENTAL RESULTS AND DISCUSSION

A. Experimental Results

The experiment uses ACC and P_{opt} to evaluate the performance of different optimal solutions selection strategies. The results are shown in the Table III and Table IV. In Table III and Table IV, the first column indicates the names of projects. The second column MULTI-M is the median value of all optimal solutions, representing the average performance of the MULTI method. The third to fifth columns represent the performance of the models based on strategies BP, CP, and CCB, respectively. Since the experiment uses 10 times 10-fold cross validation technology, each strategy will get 100 result values. All the values in the Table III and Table IV reflect the median of the 100 result values.

As we can see from Table III and Table IV, the performance of strategy BP is far superior to the performance of MULTI-M in the ACC and P_{opt} on six data sets. In addition, compared to MULTI-M, the performance of strategy CP is even worse, and the performance of strategy CCB is equivalent. Finally, we use Wilcoxon signed-rank test and Cliff's δ determine the significant difference between different optimal solutions selection strategies and MULTI-M. If and only if p-value is less than 0.5 and Cliff's δ is greater than or equal to 0.147, the performance of our strategy is significantly different from MULTI-M, otherwise the difference is negligible. We have bolded the values that have significant differences.

Conclusion. Compared with MULTI-M, the performance of strategy BP is significantly better than MULTI-M, the average performance can be increased by 12% on indicator ACC, and the average performance on indicator P_{opt} can be increased by 15%. In addition, the performance of strategy CCB is comparable to MULTI-M, and the performance of strategy

TABLE III
COMPARISON OF THREE STRATEGIES AND MULTI-M USING ACC

project	MULTI-M	BP	CP	CCB
BUG	0.633	0.774	0.251	0.636
COL	0.723	0.804	0.344	0.721
JDT	0.658	0.723	0.371	0.653
MOZ	0.577	0.606	0.213	0.579
PLA	0.682	0.746	0.179	0.683
POS	0.612	0.703	0.252	0.613
Average	0.648	0.726	0.268	0.648

TABLE IV
COMPARISON OF THREE STRATEGIES AND MULTI-M USING P_{opt}

project	MULTI-M	BP	CP	CCB
BUG	0.771	0.930	0.492	0.773
COL	0.842	0.936	0.547	0.841
JDT	0.764	0.880	0.557	0.767
MOZ	0.731	0.812	0.473	0.731
PLA	0.792	0.887	0.528	0.790
POS	0.747	0.900	0.441	0.749
Average	0.775	0.891	0.506	0.775

CP is worse than MULTI-M. Therefore, we advise to use BP-based optimal solutions selection strategy to improve the performance of the MULTI method in the effort-aware JIT-SDP problem.

B. Discussion

The experimental results show that the performance of our optimal solutions selection strategies have the following characteristics.

$$BP > CCB > CP$$

In order to explain this phenomenon, we further analyze the experimental results. Take data sets BUG as an example, the experiment uses 10 times 10-fold cross validation to perform model evaluation, so a total of 100 runs will be produced. The results of one run of the experiment is shown in the Fig.2 and Fig.3. As we can see from Fig.2 and Fig.3, each graph contains 200 red dots, each representing an optimal solution. In the Fig.2 and Fig.3, the horizontal axis represents the value of the benefit, and the vertical axis represents the value of the performance indicators (i.e., ACC and P_{opt}).

It is obvious that in an optimal solution set, the benefit values of optimal solutions are positively correlated with the performance indicators including ACC and P_{opt} . Our experimental results show that this feature is also present on five other data sets. Therefore, BP-based optimal solutions selection strategy can significantly improve the performance of MULTI.

VI. THREATS TO VALIDITY

External validity. Although the data sets used in the experiment are widely used in the field of JIT-SDP [6] [5] [4], we still cannot guarantee that the findings of the experiment will apply to all other defect data sets. Therefore, more data sets in different fields have yet to be mined and shared to verify the generalization of experimental conclusions.

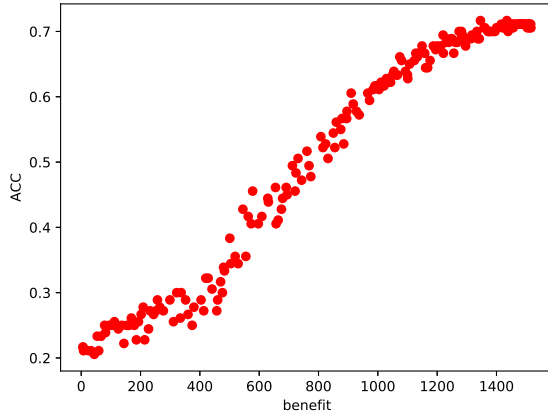


Fig. 2. ACC values in an optimal solution set

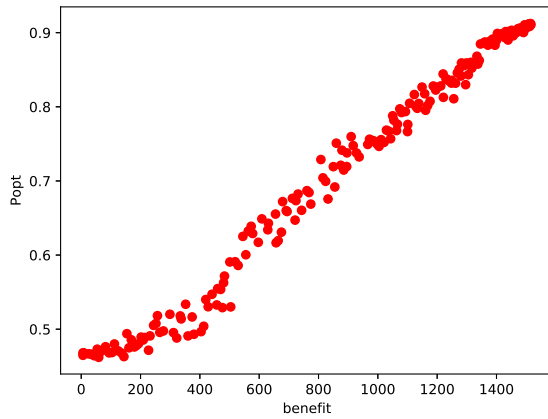


Fig. 3. P_{opt} values in an optimal solution set

Construct validity. Threats to construct validity are mainly considered whether the evaluation indicators can accurately reflect effort-aware prediction performance of models. Our experiment uses ACC and P_{opt} to evaluate effort-aware prediction performance of models, which are widely used in previous JIT-SDP studies [6] [5] [4].

Internal validity. The threats to internal validity are mainly about the accuracy of experimental code. Previous research code is mainly written in R language [5] [4], while our experimental code is written in python. In order to reduce the errors in the code, we check all the code and use the mature python libraries.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, in order to improve the performance of the MULTI method, we propose three optimal solutions selection strategies BP, CP, and CCB. In order to verify the effectiveness of the strategies, we conduct a large-scale empirical study on data sets of six open source projects. Experimental results show that compared with the average performance of MULTI,

BP-based optimal solutions selection strategy can effectively improve the performance of MULTI.

In the future, we hope to further expand our work. First, since the experiment only uses data sets from open source projects, we will collect data sets from commercial projects to further verify the generalization of experimental conclusions. Secondly, we only consider the cross-validation scenario when evaluating the performance of prediction models. In the future, we will extend the work to cross-project-validation and timewise-cross-validation scenarios to further verify the generalization of the experimental conclusions.

ACKNOWLEDGMENT

This work is partially supported by the NSF of China under grants No.61772200 and 61702334, Shanghai Pujiang Talent Program under grants No. 17PJ1401900. Shanghai Municipal Natural Science Foundation under Grants No. 17ZR1406900 and 17ZR1429700. Educational Research Fund of ECUST under Grant No. ZH1726108. The Collaborative Innovation Foundation of Shanghai Institute of Technology under Grants No. XTCX2016-20.

REFERENCES

- [1] Z. Li, X. Jing, X. Zhu, Progress on approaches to software defect prediction, *IET Software* 12 (3) (2018) 161–175.
- [2] X. Chen, Q. Gu, W. Liu, S. Liu, C. Ni, Survey of static software defect prediction, *Journal of Software* 27 (1).
- [3] S. McIntosh, Y. Kamei, Are fix-inducing changes a moving target?: a longitudinal case study of just-in-time defect prediction, in: *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018*, 2018, p. 560.
- [4] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, N. Ubayashi, A large-scale empirical study of just-in-time quality assurance, *IEEE Transactions on Software Engineering* 39 (6) (2013) 757–773.
- [5] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, H. Leung, Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models, in: *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016*, 2016, pp. 157–168.
- [6] X. Chen, Y. Zhao, Q. Wang, Z. Yuan, MULTI: multi-objective effort-aware just-in-time software defect prediction, *Information & Software Technology* 93 (2018) 1–13.
- [7] A. Mockus, D. M. Weiss, Predicting risk of software changes, *Bell Labs Technical Journal* 5 (2) (2000) 169–180.
- [8] X. Yang, D. Lo, X. Xia, J. Sun, TLEL: A two-layer ensemble learning approach for just-in-time defect prediction, *Information & Software Technology* 87 (2017) 206–220.
- [9] W. Fu, T. Menzies, Revisiting unsupervised learning for defect prediction, in: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017*, 2017, pp. 72–83.
- [10] M. Harman, S. A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, *ACM Computing Surveys* 45 (1) (2012) 11:1–11:61.
- [11] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, K. Matsumoto, An empirical comparison of model validation techniques for defect prediction models, *IEEE Transactions on Software Engineering* 43 (1) (2017) 1–18.
- [12] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation* 6 (2) (2002) 182–197.
- [13] T. Mende, R. Koschke, Effort-aware defect prediction models, in: *14th European Conference on Software Maintenance and Reengineering, CSMR*, 2010, pp. 107–116.
- [14] E. Arisholm, L. C. Briand, E. B. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, *Journal of Systems and Software* 83 (1) (2010) 2–17.

Empirical Studies Concerning the Maintenance of BPMN Diagrams: A Systematic Mapping Study

Ursula Campos, Adriana Lopes and Tayana Conte
USES Research Group
Institute of Computing, Federal University of Amazonas
Manaus, AM - Brazil
{usc, adriana, tayana}@icomp.ufam.edu.br

Simone Diniz Junqueira Barbosa
Informatics Department
PUC-Rio
Rio de Janeiro, Brazil
simone@inf.puc-rio.br

Abstract — Business process models help understand the organizational process and the software that supports it. BPMN (Business Process Modeling and Notation) is the standard notation for business process modeling, and it is widely accepted in industry. BPMN models can elucidate the activities carried out by a software during its construction and maintenance. However, during the maintenance of the software that supports an organizational process, usually only the source code of the software undergoes modifications, even when inserting new features. The software design models, including the BPMN models, often become outdated over time and, in future maintenance, they will not help understand the business process in which the software is inserted and which the software aims to support. Such scenario highlights the importance of supporting the maintenance of BPMN models. However, what has been experimentally investigated regarding the maintenance of BPMN models? To answer this question, we performed a systematic mapping, which showed experimental studies, factors and technologies that influence the maintenance of BPMN models. These results present conclusions about the state of the art and gaps that can be explored in this field of research.

Keywords - *Business Process Model and Notation; BPMN; Software Maintenance; Model Maintenance; Systematic mapping.*

I. INTRODUCTION

Business process modeling is an essential activity for the success of business process management (BPM) [1]. The BPMN (Business Process Modeling and Notation) is the standard notation maintained by OMG [2] for business process modeling and is widely accepted in industry [3]. Researchers and practitioners recognize that understanding the business processes is key to identifying the users' needs that a software should support [4]. Martinez et al. [5] emphasize that the focus on software development from business processes can increase the software's compliance with the needs of its users. BPMN models can help in understanding the activities carried out by the software and better support software maintenance.

Maintenance is one of the most crucial phases of the software life cycle. It is usually divided into two steps: understanding the software artifact and modifying the software artifact [6]. In fact, a software artifact should be well understood before being modified, because developers need to know what impact modifications will have on the software and, possibly, on the underlying business processes. In the maintenance stage, process

models are of great relevance in understanding the software being developed [7] or modified. Thus, it is important that they are consistent with the current version of the software, so that it is understood correctly.

However, during the software maintenance process, usually only the source code is modified, even when inserting new features. Over time, development team turnovers mean that new members may not know the software functionalities, since they did not participate in its design and development [8]. It is in this scenario that software models prove to be of great value, because developers first seek to understand the software functionalities through the models built during its project. However, the documentation is almost always outdated and inconsistent [9]. When the models are not maintained together with the software, the information in these models will be inconsistent with the current version of the software, which hinders new professionals' understanding and activities during software maintenance.

As we realize the importance of supporting the maintenance of BPMN models, it is important to know what has been experimentally investigated on the maintenance of BPMN models. The goal of this paper is to describe a systematic mapping of studies related to the maintenance of BPMN models in order to identify what has been done in the literature to support and facilitate the maintenance of such models. The systematic mapping revealed factors and technologies related to the context of maintenance of BPMN models. With this work we present conclusions about the state of the art and gaps that can be explored in this field of research.

The remainder of this paper is organized as follows: Section 2 presents the related works. Section 3 describes the methodology applied to conduct the systematic mapping. Section 4 presents the results of the mapping study. Section 5 discusses the threats to the validity of this work. Finally, Section 6 presents the conclusions of the systematic mapping.

II. RELATED WORKS

In this section, we present some related works to our work. Pourmirza et al. [10] present a systematic literature review of Business Process Management Systems (BPMS) architectures. BPMS are information systems that interpret business processes to ensure that the activities specified therein are properly

executed and monitored by an organization. In this work, BPMS architectures that served as primary studies were compared with respect to the reference architecture that they are based on, the level of detail at which they are described, the architectural styles that they use, the means with which they are evaluated, and the functionality that they support. The resulting comparison provides an overview of and insights into the current body of knowledge on BPMS architectures.

Valença et al. [11] present a systematic mapping study of business process variability approaches, which is an emergent field in BPM with many of its proposals inspired by theories from Software Product Line to handle process variability. According to the authors, variability in business processes is necessary for organizations dealing with environmental changes. The results show that a significant number of approaches is available, but most of them lack empirical studies.

Although these works deal with BPM, they do not exploit the research done on the maintenance of BPMN models. Our work investigates the maintenance of BPMN models in the literature.

III. RESEARCH METHOD

A systematic mapping is a broad review of primary studies in a specific topic area, aiming to identify what evidence is available on the topic [12]. We followed the guidelines proposed by Kitchenham and Charters [12]. The following subsections detail our systematic mapping protocol.

A. Goal

We had the following goals for the systematic mapping:

- To investigate the maintenance of BPMN models and whether the models are understandable and modifiable to allow them to be modified while maintaining the source code.
- To gather experimental evidence on the use of BPMN models in their maintenance or use during the maintenance of the software source code.

B. Research Question

This mapping aimed to answer the following research question: ***“What has been experimentally investigated regarding the maintenance of BPMN models?”***. In order to answer this question, we divided the systematic mapping into specific sub-questions about the maintenance of BPMN models (see Table I).

TABLE I. RESEARCH SUBQUESTS

Subquestion	Description of Ssubquestion
SQ1	What is the state of the art in experimental studies on maintenance of BPMN models or source code maintenance when using BPMN models?
SQ2	Which dependent variables are investigated in the experimental studies?
SQ3 ¹	Which of the factors studied influence the software maintenance capability (source code or model)?
SQ4	What technologies support the maintenance of BPMN models?

¹ We considered as factors the results of the publications selected in the systematic mapping that influence the maintenance of BPMN models.

² <http://www.scopus.com>

C. Search strategy

To construct the search string, we defined the search terms based on the procedure described by Kitchenham and Charters [12], who suggested defining the parameters for Population, Intervention, Comparison, Result, and Context (PICOC). The population was the specific field of research on BPMN - Business Process Modeling and Notation; the intervention was composed of maintenance phases or types; the result was the types of experimental studies; comparison and context were not applicable because our goal is to characterize what is done in relation to the maintenance of BPMN models, so there is no comparison to determine the context. We have identified the terms of the research through the publication of Fernandez-Saez et al. [6], which we used as the basis for this work. They investigated the maintenance of UML models, so we adapted this work by bringing it into the context of BPMN models. Table II shows the terms used in the search string and the groups of synonyms used in its construction. We used the boolean operator OR between the alternative terms and synonyms, and the Boolean operator AND to join the groups.

We used the search string in the Scopus², Engineering Village³ and ACM⁴ digital libraries. We have included the Scopus and Engineering Village libraries because they are meta libraries and index publications from several reputable publishers in Software Engineering, such as ACM, IEEE, Springer and Elsevier, and they allow defining filters by type of document, language and area of knowledge. Although the ACM library is indexed by Scopus, we included this library to ensure that there were no excluded publications in the Scopus indexing and because ACM indexes some Springer Link and Science Direct publications as well.

TABLE II. SEARCH STRING TERMS AND SYNONYMS

Term of PICOC	Main Term	Synonyms
Population	BPMN	business process model and notation OR business process modeling and notation
Intervention	Maintenance	evolution OR comprehension OR maintainability OR evolvability OR understandability OR modularity OR modification OR understanding OR reusability OR stability OR misinterpretation OR analyzability OR testability OR changeability OR comprehensibility
Results	Empirical	survey OR action research OR experiment OR case study

D. Selection Criteria

The selection process comprised three steps (first filter, second filter and process of extracting data). In the first filter, two researchers only read the title and the abstract. They have selected the publications applying the inclusion and exclusion criteria (see Table III). The reliability of the inclusion and

³ <http://www.engineeringvillage.com>

⁴ <http://dl.acm.org>

exclusion criteria of a publication in the systematic mapping was assessed by applying Fleiss' Kappa [13]. Fleiss' Kappa is a statistical measure for assessing the reliability of agreement between a fixed number of raters when classifying items.

TABLE III. INCLUSION AND EXCLUSION CRITERIA

#	Selection Criteria
IC1	Papers with experimental studies using BPMN model during the maintenance of the diagram or source code
IC2	Papers with experimental studies with BPMN models helping in the process of understanding the software
IC3	Papers that evaluate the comprehensibility or maintainability of BPMN diagrams
#	Exclusion Criteria
EC1	Papers proposing BPMN extensions;
EC2	Papers that do not report experimental studies;
EC3	Papers that mention BPMN or maintenance only as general introductory terms in the abstract and nowhere else;
EC4	Papers that are not written in English or Portuguese;
EC5	Papers unavailable for reading or data collection (paid publications, broken links in the search engine and not made available by the authors after an attempt to contact);
EC6	Duplicate papers

Initially, we asked two researchers to classify, individually, a random sample of 20 publications to analyze the degree of agreement in the selection process through the Fleiss' Kappa [13]. The selected sample was the set of the publications returned by Scopus. The result of the degree of agreement showed a substantial level of agreement between the two researchers ($Kappa = 0.653$).

In the second filter, the researchers fully read the selected publications, selecting them according to the same criteria used in the first filter. After completing the selection process, we started the process of extracting data.

We used an extraction form to standardize the data collected. According to Fernandez et al. [6], this ensures that the same criteria will be used, thus facilitating their classification. We extracted the data according to each subquestion. The complete protocol and the publications obtained in the second filter are available in a technical report [14].

IV. RESULTS

Figure 1. depicts the publications selection process carried out in the conduction of the systematic mapping. The search string returned 89 publications in Scopus library, 80 in Engineering Village, and 19 in ACM. After eliminating duplicates, we had 88 publications in Scopus, 7 in Engineering Village and 15 in ACM, resulting in 110 publications. During the first filter, we rejected 81 publications that did not meet the inclusion criteria. We read the remaining 29 publications in full and classified them in the second filter. At that stage we selected 18 publications that proceeded to the extraction process.

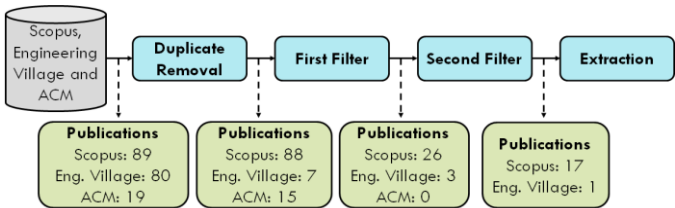


Figure 1. Article selection process.

The selected papers were published between 2008 and 2018. There was no criterion to limit the year of publications. Figure 2. shows that most publications are recent, which leads us to believe that the maintenance of BPMN models is a timely research issue.

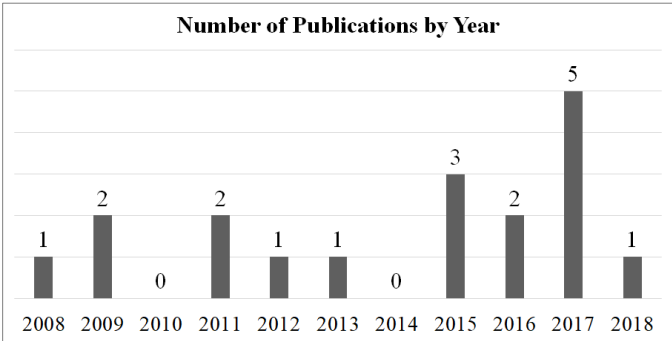


Figure 2. Overview of the publications per year.

The answers we found for each research sub-question are the following:

1) *SQ1: What is the state of the art in experimental studies on maintenance of BPMN models or source code maintenance when using BPMN models?*

This subsection presents several items related to the state of the art of experimental studies regarding the maintenance of BPMN models. We present these items next:

Experimental Study Type: Figure 3. shows the number of publications with each type of experimental study. Even when a publication reported more than one controlled experiment, i.e., replications of the experiment in other institutions or with other artifacts, Figure 3. only counts it once.

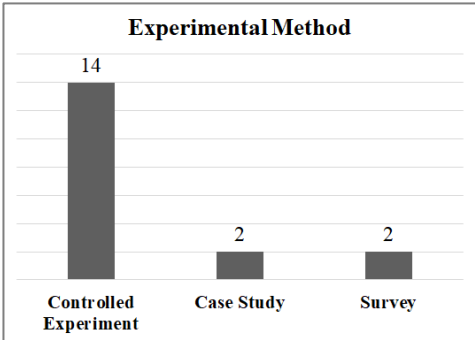


Figure 3. Types of experimental studies reported in the publications.

Experimental studies: When counting the number of experimental studies reported in the 18 publications, according to each type of experiment. If one publication reported two

controlled experiments, we counted the two experiments. In total, the 18 publications presented 39 experimental studies: 35 controlled experiments, 2 case studies, and 2 surveys.

Context: We classified the context of the studies as laboratory, industry or online. Most publications (72.22%) reported experiments that were conducted in a laboratory within academic environment. Two publications (11.11%) reported experiments carried out in industry (in companies that use BPMN models). Three publications (16.67%) reported experiments performed online, where a task was made available online and the participant had a deadline to do it (e.g., a week). **We can see that the amount of experiments performed in industry is low**, which corroborates the result already obtained in the previous section, about the need to perform experiments in real environments.

Characterization of the participants: Most publications have presented experimental studies conducted with undergraduate or graduate students, which is not necessarily inappropriate, since student skills are considered similar to the skills of novice practitioners [15][16]. Some publications have conducted experiments with more than one participant profile. **Among the 39 experiments presented in the publications, 29 were experiments with only students as subjects**, 5 were experiments with only professionals, 3 were experiments with students and professionals, 1 experiment with students, professionals and academic professionals, and 1 experiment in which the participant was the author of the technique. In the latter, the author applied the technique proposed by himself, to reduce the complexity of BPMN models.

Maintenance focus object: Software maintenance tasks usually require some modification in the source code, and should be accompanied by modifications in the corresponding BPMN model. **We did not find experimental studies that reported the use of BPMN models during source code maintenance.** We only found publications where the maintenance was done only in the BPMN model, consisting mostly of experimental studies that evaluate the comprehensibility of BPMN models.

Treatments in experimental studies: Figure 4. shows the different treatments used in the experimental studies. We divided the treatments into six categories: (i) complexity of the model, (ii) model representation, (iii) model characteristics, (iv) type of model representation, (v) model representation method, and (vi) characteristics of the model maintainers and defects in the models.

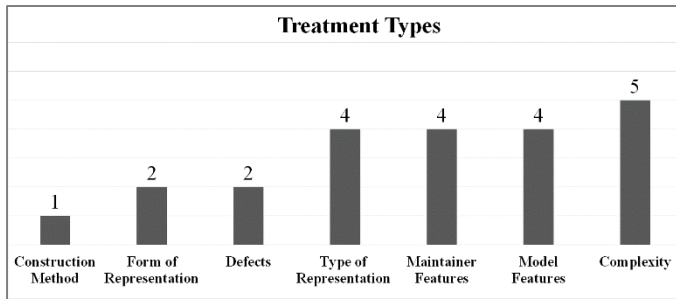


Figure 4. Types of treatment in experiments, by publication.

We highlight the category *complexity of the model* with the highest number of publications. **Some publications presented experiments with more than one type of treatment**, thus the bars add up to 22 publications. The types of treatment of each publication are more detailed in a technical report [14].

According to the data obtained, we can conclude the following about the state of the art of research on the maintenance of BPMN models:

Most of the **experimental studies carried out** were controlled experiments, carried out in the laboratory, with the participation of students. Various **types of treatment** were applied such as: the complexity of the models, the form of representation, the characteristics of the models, among others. The **focus of the maintenance** was always the model itself – not coupled with source code maintenance.

2) *SQ2: Which dependent variables are investigated in the experimental studies?*

The dependent variables we investigated in the experimental studies are represented in Figure 5. We identified four dependent variables: model understanding, model modifiability, model complexity, and model completeness (i.e., to what extent the model represented the functionalities of a specific process). We can see that the great majority of publications sought to evaluate the comprehension of BPMN models (17), some publications evaluated the modifiability (4) and complexity (4) of the models and only one publication evaluated the completeness of the BPMN models. Some papers investigated both the comprehension and the modifiability or complexity of BMPN models in the same experiment, so the bars add up to 26.

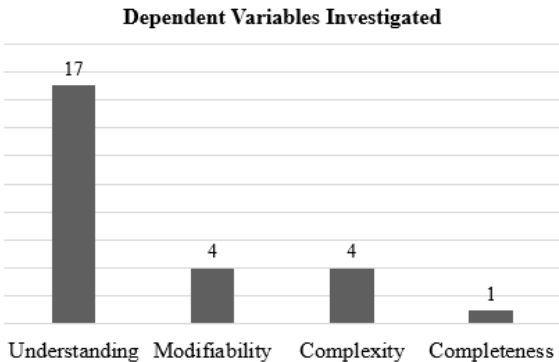


Figure 5. Dependent variables investigated.

3) *SQ3: Which of the factors studied influence the software maintenance capability (source code or model)?*

As we have already mentioned, we have not found experimental studies that dealt with the maintenance of the source code and BPMN together. This research subquestion will therefore focus only on the model. Figure 6. shows the factors that influence, either positively or negatively, the maintenance of BPMN models, according to the results obtained in the experiments of the selected publications (publications are called PB#, such as PB1 for the first publication in the report [14]).

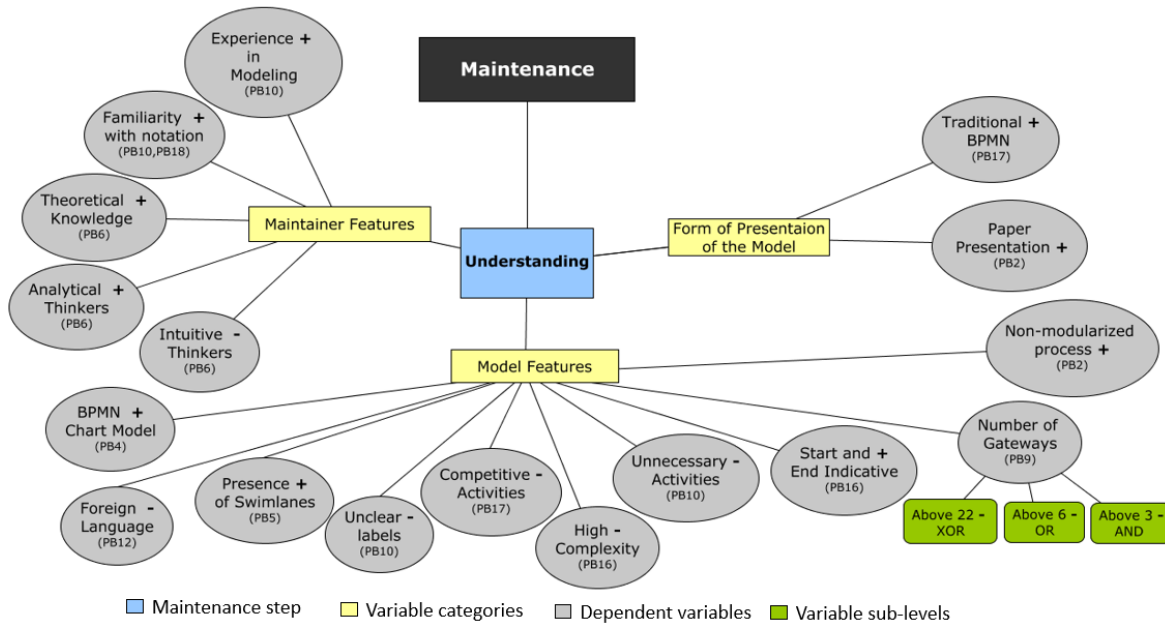


Figure 6. Dependent variables per publication.

It is important to remember that the maintenance activity is divided into two tasks [6]: artifact understanding and artifact modification. Considering these two types of maintenance tasks, the factors identified in the results of the experiments were related to *understanding* (see Figure 6.). The experiments that dealt with modifying the models actually had some kind of technology to support the modification and were therefore fitted as a response to SQ4. Factors that did not present significant results (if the study could not determine whether a factor influenced or not the understanding of a BPMN model) were omitted. We classified as positive (+) the factors that facilitate the understanding of the model and as negative (–) the factors that undermine the understanding of the model.

4) SQ4: What technologies support the maintenance of BPMN models?

We divided the technologies that support the maintenance of models according to the two types of maintenance tasks: understanding and modification. We then evaluated whether the proposed technology supported understanding or modifying the model during maintenance. Figure 7. shows the technologies identified to support the maintenance of BPMN models.

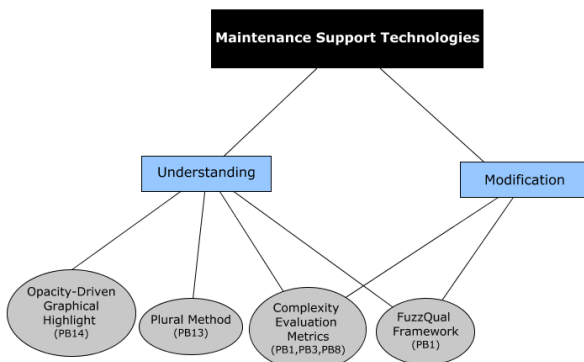


Figure 7. Technologies that support the maintenance of BPMN models

Technologies aimed at supporting the maintenance of BPMN models that focused on understanding were as follows:

- **Opacity-Driven Graphical Highlights Technique:** Jost and Hericko's approach [17] makes business process models less complex without changing the notation. The technique consists of working on the opacity of the models, highlighting only the relevant parts of the model according to the context in which it is used.
- **Plural Method:** decentralized method for creating BPMN models, in which the sources of model construction are also participants in the process. The method aims to involve process participants in modeling, since they are well aware of the problem domain.

Technologies to support the maintenance of BPMN models that focused on understanding and modification were as follows:

- **Complexity measurement metrics:** measures that may be useful in forecasting different aspects of understanding and modifying business process models in future model maintenance. The goal is to make models easier to understand and modify for all stakeholders.
- **FuzzQual Framework:** approach that evaluates BPMN models with regard to comprehensibility and modifiability. A framework/system built in JAVA evaluates the model based on the complexity metrics chosen, among several metrics that were evaluated. Through fuzzy logic, the framework evaluates the model with classifications of the type: "Moderately difficult to understand with a degree of certainty of 63%", "Moderately difficult to modify with a degree of certainty of 100%". This framework is an implementation of the complexity metrics to evaluate BPMN models.

V. THREATS TO VALIDITY

Although we conducted this research under a systematic mapping methodology by defining a research protocol, some

threats to validity can be identified: (i) the researchers' bias regarding the analysis of the primary studies; (ii) the university's limited access to some scientific databases, which can prevent some publications from being accessed; (iii) the limitation of the scope of this research to the selected databases. These threats were minimized by taking some actions. For the first threat, we reviewed the review protocol and performed the Kohen's Kappa statistical test in order to reduce the researchers' bias. Additionally, another experienced researcher reviewed the execution process. For the second threat, we had two publications that fit that threat. We requested the authors for the full publication whenever possible and included those that have been made available. Regarding the third threat, although the research was conducted in only three databases, they index publications from a large number of well-known publishers, journals and conferences.

VI. CONCLUSIONS

This work describes the results of the systematic mapping of the literature that we carried out to identify what has been experimentally investigated on the maintenance of BPMN models. The main conclusions we have reached with the systematic mapping can be summarized as follows:

- Most studies were carried out in academic environments, which is explained by the fact that they are the most accessible environment for researchers. The difficulty to perform research in real environments, within industry/companies, is well known. However, there is a need for further experiments in real environments.
- Most studies performed are controlled experiments. This demonstrates a need for more case studies to confirm the results obtained in real environments.
- We have not identified studies that explored the maintenance of BPMN models together with maintenance of the software itself. We have also not identified studies that investigate the impact of updated or outdated BPMN models during software maintenance.
- The focus of the experimental studies is almost entirely on the *understanding* of BPMN models. However, understanding is only one of the tasks of maintenance, and it is necessary to focus also on *modifying* the models themselves.
- The technologies that we have found focus mostly on the initial construction of the model and how it will be easily understood or modified when it is necessary to use it or modify it. The proposed technologies seek to measure in advance whether the model created will be easy to maintain in the future. However, it is also necessary to create technologies that directly support the maintenance phase of the model.

The results indicate that there is a lack of maintenance of BPMN models, especially the maintenance of these models in conjunction with maintenance of the software source code. In relation to the technologies proposed, most of the research identified focuses on understanding the models, which is the initial task of maintenance, highlighting the need for technologies that support the modification of BPMN models.

This shows research opportunities to be explored in this field of research in future work.

ACKNOWLEDGMENTS

We thank all the students who participated in this research. We also thank the financial support granted by CAPES process 175956/2013, FAPEAM through Edital 009/2017; CNPq processes 311494/2017-0, 430642/2016-4, 423149/2016-4, and 311316/2018-2.

REFERENCES

- [1] O. Turetken, T. Tompen, I. Vanderfeesten, A. Dikici and J. Van Moll, "The effect of modularity representation and presentation medium on the understandability of business process models in BPMN", Lecture Notes in Computer Science, vol. 9850, Springer, Cham, 2016.
- [2] OMG, "Business process model and notation (BPMN)", v. 2.0, 2011.
- [3] Z. Bukhsh, Z. Sinderen, N. Sikkil, and D. Quartel, "Understanding Modeling Requirements of Unstructured Business Processes", 2017, pp. 17-27.
- [4] Shishkov, B., Xie, Z., Liu, K., and Dietz, J. L. (2002). "Using norm analysis to derive use cases from business processes". In Proceedings of the 5th Workshop On Organizational Semiotics.
- [5] A. Martinez, J. Castro, O. Pastor, H. Estrada, "Closing the gap between organizational modeling and information system modeling". Workshop de Engenharia de Requisitos, Piracicaba, São Paulo-SP, 2003, pp. 93-108.
- [6] A. Fernández-Sáez, M. Genero, M. R. V. Chaudron, "Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study", Information and Software Technology 55, 2013, pp. 1119-1142.
- [7] E. Cruz, R. Machado, M. Santos, "Bridging the Gap between a set of interrelated Business Process Models and Software Models", 17th International Conference on Enterprise Information Systems (ICEIS), 2015, pp.338 – 345.
- [8] E. Arisholm, L.C. Briand, S.E.Hove, Y. Labiche, "The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation", IEEE Transactions on Software Engineering v. 32, 2006, pp.365-381.
- [9] A. Forward, T. Lethbridge, "The relevance of software documentation, tools and technologies: a Survey", In Proceedings of the 2002 ACM symposium on Document engineering, 2002, pp. 26-33.
- [10] S. Pourmirza, S. Peters, R. Dijkman, and P. Grefen. A systematic literature review on the architecture of business process management systems. Information Systems, 66, 2017, pp.43-58.
- [11] G. Valença, Alves, C. Alves, V. and N. Niu, "A systematic mapping study on business process variability", International Journal of Computer Science & Information Technology, 5(1), 2013, p.1
- [12] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering", EBSE Technical Report – EBSE 2007, Durham.
- [13] J.L. Fleiss, "Statistical Methods for Rates and Proportions", Second ed., John Wiley & Sons, New York, 1981..
- [14] U. Campos, A. Lopes, S. de Souza, T. Conte, , "A systematic mapping on Empirical Studies Concerning the Maintenance of BPMN Diagrams", TR-USES-2019-0002, 2019. Available online at: <http://uses.icomp.ufam.edu.br/wp-content/uploads/2019/03/TR-USES-2019-002.pdf>
- [15] D. Budgen, A. Burn, P. Brereton, B. Kitchenham, R. Pretorius, "Empirical evidence about the UML: a systematic literature review", Software: Practice and Experience, 2010.
- [16] P. Brereton, B. Kitchenham, D. Budgen, M. Turner, M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain", Journal of Systems and Software 80, 2007, pp. 571–583.
- [17] G. Jošt, M. Heričko and G. Polančič, Softw Syst Model, 2017. <https://doi.org/10.1007/s10270-017-0618-5>

Multistep Flow Prediction on Car-Sharing Systems: A Multi-Graph Convolutional Neural Network with Attention Mechanism

Yi Luo^{*†}, Qin Liu^{*†‡}, Hongming Zhu^{†‡}, Hongfei Fan^{†‡}, Tianyou Song[†], Chang Wu Yu[§] and Bowen Du[¶]

[†]School of Software Engineering, Tongji University

[‡]Tsingtao Advanced Research Institute, Tongji University

[§]Department of Computer Science and Information Engineering, Chung Hua University

[¶]Department of Computer Science, University of Warwick

Email: {1731530, qin.liu, zhu_hongming, fanhongfei, 1551177}@tongji.edu.cn, cwyu@chu.edu.tw, B.Du@warwick.ac.uk

Abstract—Multistep flow prediction is an essential task for the car-sharing systems. An accurate flow prediction model can help system operators to pre-allocate the cars to meet the demand of users. However, this task is challenging due to the complex spatial and temporal relations among stations. Existing works only considered temporal relations (e.g., using LSTM) or spatial relations (e.g., using CNN) independently. In this paper, we propose an attention multi-graph convolutional sequence-to-sequence model (AMGC-Seq2Seq), which is a novel deep learning model for multistep flow prediction. The proposed model uses the encoder-decoder architecture, wherein the encoder part, spatial and temporal relations are encoded simultaneously. Then the encoded information is passed to the decoder to generate multistep outputs. In this work, specific multiple graphs are constructed to reflect spatial relations from different aspects, and we model them by using the proposed multi-graph convolution. Attention mechanism is also used to capture the important relations from previous information. Experiments on a large-scale real-world car-sharing dataset demonstrate the effectiveness of our approach over state-of-the-art methods.

Index Terms—Car-sharing systems, Multistep flow prediction, Graph convolution network

I. INTRODUCTION

In recent years, car-sharing systems have been introduced to a number of cities as a means of increasing mobility, reducing congestion, and pollution [1]. Car-sharing systems involve a small to medium fleet of cars, which are available at several stations, to be used by a relatively large group of users. Users can pick up a car in a station and drop off it at another station, which is called the one-way system [2]. The key to success of the car-sharing systems is an accurate flow prediction model, which plays a vital role in various tasks such as car rebalancing [3]. Furthermore, instead of predicting only for the next step (e.g., next day), the multistep flow prediction is more attractive to the system operators since it offers information on a long-term trend.

Traditional time series prediction methods like ARIMA have been widely used for traffic prediction problem [4, 5]. However, these approaches are often applied to a single station

separately and ignore the spatial relations with each other. For example, if a station near a railway station has high flow, another one close to it may also have high flow. Furthermore, building a separate prediction model for each station is time-consuming and impractical if there are hundreds of stations. Hence, the key challenge for this problem lies in how to model complex spatial relations and temporal dynamics. To tackle the above challenges, we propose a novel deep learning model, named attention multi-graph convolutional sequence-to-sequence model (AMGC-Seq2Seq), which captures spatial-temporal relations effectively for station-level flow prediction.

In this work, a new multistep flow prediction model for car-sharing systems is designed. Multiple graphs among stations are defined to represent their heterogeneous spatial relations. Then we employ the proposed multi-graph convolution to model these spatial correlations. Furthermore, a novel deep learning framework, named AMGC-Seq2Seq, is proposed to capture the spatial and temporal relations simultaneously by incorporating the encoder-decoder architecture with graph convolution networks. The proposed method is validated on a large-scale real-world car-sharing dataset from EVCARD. The dataset contains car orders through EVCARD service in the city of Shanghai in China over three months, with about 480,000 orders per month on average. We conducted extensive experiments to compare with state-of-the-art methods and have demonstrated the superior performance of our proposed method.

The rest of this paper is organized as follows: Section II reviews the existing works. Section III first formulates the multistep flow prediction problem and then describes the details of the proposed AMGC-Seq2Seq model. Section IV presents the experiment settings and discusses the obtained results. Section V concludes this work.

II. RELATED WORK

The problem of car-sharing system flow prediction is similar to traffic prediction problem, of which the goal is to predict the traffic-related value (e.g., traffic flow or traffic speed) for a period of time through historical data. A number of studies

^{*}These authors contributed equally to this work

have investigated traffic prediction for decades. In this section, we discuss the related work on traffic prediction problem.

The early research on traffic prediction focused on the prediction of the individual station using classical empirical statistical methods. Among all the traditional methods, the autoregressive integrated moving average (ARIMA) and its variants are the most widely used [6]. Based on this time series model, recent studies also consider adding external context data, such as weather, wind speed and event information [7]. Besides, various techniques have been used to model spatial interactions. Deng et al. [8] apply non-negative matrix factorization on road networks to capture correlations between roads. Tong et al. [9] mainly adopt POI data as the spatial features. However, all of these methods are based on the time series model and ignore complex spatial-temporal relations.

With the success in deep learning, more and more researchers attempt to use deep learning techniques on traffic prediction problem. Zhao et al. [10] use the long short-term memory (LSTM) networks to capture non-linear temporal relations. Wang et al. [11] propose the DeepSD, which utilizes multiple data sources and predict the gap between the car-hailing supply and demand. These methods focus on temporal features extraction but do not model the spatial-temporal relations.

To effectively model the complex spatial relations, some researchers use convolutional neural network (CNN) to capture adjacent relations among the traffic networks. Yao et al. [12] propose the DMVST-NET, which models both spatial and temporal relations by local CNN and LSTM. Since the traffic networks are naturally non-Euclidean as the data format is no longer a matrix and CNN becomes less helpful, some researchers turn to use graph convolutional network (GCN) to model this non-Euclidean structures. Chai et al. [13] propose a multi-graph convolutional network to catch heterogeneous inter-station spatial correlations. However, all of these methods are proposed for one-step prediction. Specifically, for multistep prediction, the output from the previous step is taken as the input to the current step, which usually leads to error accumulation and poor prediction performance.

Several researchers have recently attempted to investigate multistep prediction. Cai et al. [14] propose an improved KNN model to achieve multistep forecasting. They describe the traffic state of a road segment by a spatial-temporal state matrix and use the Gaussian weighted Euclidean distance to measure the similarity. Park et al. [15] propose the AGC-Seq2Seq for multistep speed prediction, which learns the spatial-temporal relations simultaneously by integrating LSTM and GCN. They utilize encoder-decoder to model the multistep prediction problem. However, they use only one graph to model the spatial relations, which may not be enough to reflect the complex spatial relations, and they have not capture the local temporal relations for the individual station.

Inspired by those research accomplishments and the real-world problem observations, a potential solution for more accurate and practical prediction should be an integrated analysis for both spatial and temporal relations of stations.

III. PROPOSED MODEL

In this section, we first formulate the problem and then describe how to model the spatial and temporal relations using the proposed attention multi-graph convolutional sequence-to-sequence model (AMGC-Seq2Seq).

A. Problem Formulation

In car-sharing systems, there are two types of flows: outflow and inflow. The outflow of station i is defined as the pick-up frequency at the time slot t (e.g. one day), which is denoted by $y_{i,t}^{out}$. The inflow of station i is defined as the drop-off frequency at the time slot t , which is denoted by $y_{i,t}^{in}$.

Suppose we have N stations, then the outflow of all stations at time slot t can be denoted as $Y_t^{out} = [y_{1,t}^{out}, y_{2,t}^{out}, \dots, y_{N,t}^{out}]$, and inflow of all stations at time slot t can be denoted as $Y_t^{in} = [y_{1,t}^{in}, y_{2,t}^{in}, \dots, y_{N,t}^{in}]$.

Suppose the current time slot is t , and we have the historical data $[(Y_1^{in}, Y_1^{out}), (Y_2^{in}, Y_2^{out}), \dots, (Y_t^{in}, Y_t^{out})]$, the problem considered in this work is to predict the flow at next T steps $[(\hat{Y}_{t+1}^{in}, \hat{Y}_{t+1}^{out}), (\hat{Y}_{t+2}^{in}, \hat{Y}_{t+2}^{out}), \dots, (\hat{Y}_{t+T}^{in}, \hat{Y}_{t+T}^{out})]$, aiming to:

$$\min \sum_{k=1}^T \left\| \hat{Y}_{t+k}^{in} - Y_{t+k}^{in} \right\|_2^2, \min \sum_{k=1}^T \left\| \hat{Y}_{t+k}^{out} - Y_{t+k}^{out} \right\|_2^2 \quad (1)$$

B. Framework Overview

Figure 1 shows the architecture of our proposed model. Generally, our model uses the encoder-decoder architecture. In the encoder, we form a "hamburger" structure with two LSTMs and one multi-graph convolution layer (M-GCN) in between to model the spatial and temporal relations. First, LSTM is used to model local temporal information for each station. After that, multi-graph convolution is used to model heterogeneous spatial relations among stations. Finally, another LSTM is used to aggregate spatial-temporal relations together. Then the encoded information is passed to the decoder and incorporated with attention mechanism to generate multistep outputs. The details of each module are described as follows.

C. Encoder

1) *Temporal relations modeling*: Since the flow pattern for each station varies a lot, we adopt Long Short-Term Memory (LSTM) network [16] to model this local temporal relation. At each time step t , LSTM takes two inputs: memory of the last time step h_{t-1} and the related information at current time step x_t . Based on these inputs, LSTM learns to remove or add new information to the memory, and finally generates a new memory state h_t which accumulates all the previous information. This process is controlled by three gates: forget gate, input gate, and the output gate, which can be formulated as follows:

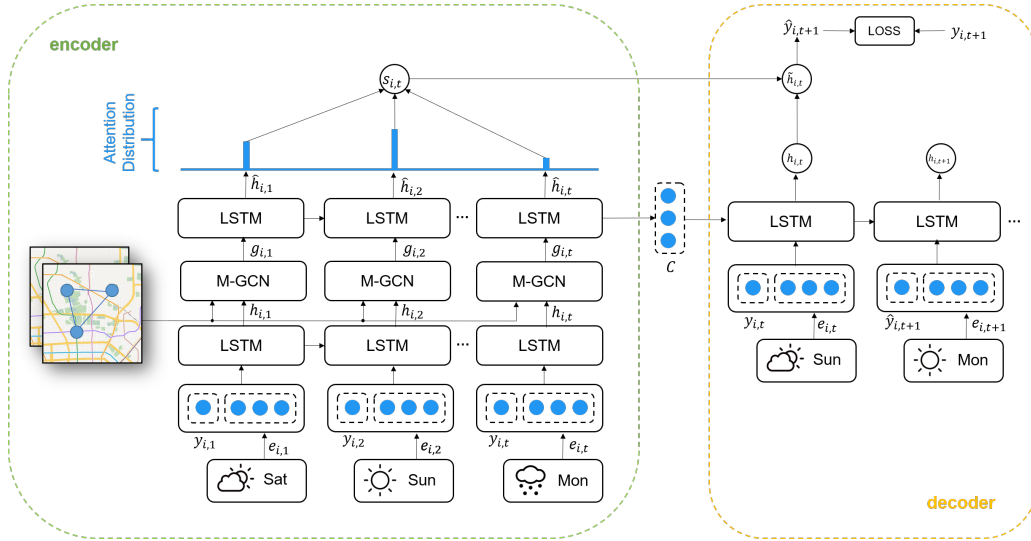


Fig. 1. The architecture of AMGC-Seq2Seq.

$$f_{i,t} = \sigma(W_f[h_{i,t-1}, x_{i,t}] + b_f) \quad (2)$$

$$i_{i,t} = \sigma(W_i[h_{i,t-1}, x_{i,t}] + b_i) \quad (3)$$

$$\tilde{C}_{i,t} = \tanh(W_C[h_{i,t-1}, x_{i,t}] + b_c) \quad (4)$$

$$C_{i,t} = f_{i,t} \circ C_{i,t-1} + i_{i,t} \circ \tilde{C}_{i,t} \quad (5)$$

$$o_{i,t} = \sigma(W_o[h_{i,t-1}, x_{i,t}] + b_o) \quad (6)$$

$$h_{i,t} = o_{i,t} \circ \tanh(C_{i,t}) \quad (7)$$

Where \circ denotes the Hadamard product. $f_{i,t}$, $i_{i,t}$ and $o_{i,t}$ are the forget gate, the input gate and the output gate respectively. σ and \tanh are the nonlinear activation functions. W_f , W_i and W_o are all trainable parameters, while b_f , b_i and b_o are the corresponding bias vectors.

As for the inputs of step t , we concatenate flow $y_{i,t}$ with external features $e_{i,t}$ (e.g., weather, weekday/weekend) together:

$$x_{i,t} = y_{i,t} \oplus e_{i,t} \quad (8)$$

It should be noted that all stations share the same weights of LSTM in the proposed model. The reason is that sharing LSTM among all stations may encourage the desired model becomes more general and reduce complexity.

2) *Spatial relations modeling*: To capture the spatial relations between stations, we propose a multi-graph convolution layer. The goal of the multi-graph convolution layer is to learn a function of features on graphs. Here we define the graphs which will be used later.

The car-sharing systems can be modeled as a weighted undirected graph, of which a node represents one station and an edge represents the relation between two stations. Usually, the large the weight of an edge is, the strong correlations there are between two stations. The simplest graph is the distance

graph, where the weight of an edge is defined as the reciprocal of the distance. In addition to the distance graph, there can be more graphs used to model the relation of stations. In this work, we propose and define the following two graphs: distance graph and POI graph.

Distance Graph: According to our observation, stations in the same area are likely to have similar flows. Therefore, the edge in the distance graph between two stations is defined to be the reciprocal of the distance.

$$A_{dis,i,j} = \begin{cases} d_{i,j}^{-1}, & i \neq j \\ 0, & i = j \end{cases} \quad (9)$$

Where $d_{i,j}$ is the distance between station i and j .

POI Graph: Intuitively, stations sharing similar functionality may have similar flows. For example, stations in the commercial area usually have more flows on weekends, and stations in the office area are expected to have more flows on weekdays. Point of interest, or POI, is a specific point location that someone may find useful or interesting (e.g., schools, shops, post offices are all POIs). Therefore, we can use POIs around a station to represent its functionality. Accordingly, we define the edge in a POI graph between two stations as the cosine similarity of POIs.

$$A_{dis,i,j} = \begin{cases} \frac{P_i \cdot P_j}{\|P_i\| \|P_j\|}, & i \neq j \\ 0, & i = j \end{cases} \quad (10)$$

Here P_i and P_j are the POI vectors of station i and j respectively. The dimension of the vector is the category of POIs, and the value in the vector is the number of the specific POI category around the station.

Recall that in the temporal modeling, we have the hidden state $h_{i,t}$, which contains the temporal information of station i at t time step. Here we define $H_t = [h_t^1, h_t^2, \dots, h_t^N]$, which

represents the temporal information of all stations at the time t . Then with the above graphs constructed, we propose the multi-graph convolution to model the spatial relations as defined in eq.(11).

$$G_t = \sigma\left(\sum_{A \in \mathcal{A}} A * H_t * W\right) \quad (11)$$

Where \mathcal{A} is the set of graphs, H is the feature matrix of all stations, and W is a trainable matrix which will be updated during the training. Here σ is a non-linear activation function, which is ReLU in our model.

Eq.(11) indicates that for each station, we update its feature by a weighted sum of the features of all the other stations. The larger the weight of the edge is between two stations, the more the feature of that station contributes. However, there are some problems need to be resolved.

First of all, according to the definition of the graph, the diagonal of the graph matrix contains all zeros. As a result, if we multiply it with the feature matrix, the vector of itself contributes nothing, which loses a lot of important information of itself. We fix this by adding an identity matrix to A .

Another problem is that since we combine multiple graphs by adding the transformed feature matrices together, the obtained graph matrices may vary a lot. Hence, we normalize A by dividing each value by the row sum such that all rows sum to one.

Therefore, we modify the eq.(11) as follows:

$$D_{i,j} = \begin{cases} \sum_{k=1}^N A_{i,k}, & i = j \\ 0, & i \neq j \end{cases} \quad (12)$$

$$\hat{A} = D^{-1}A + I \quad (13)$$

$$G_t = \sigma\left(\sum_{\hat{A} \in \hat{\mathcal{A}}} \hat{A} * H_t * W\right) \quad (14)$$

Where D is a diagonal matrix of which the value in the main diagonal is the row sum of A . Multiplying D^{-1} with A makes all rows of A sum to one. Finally, we add it with the identity matrix I to ensure self-loops in the graph.

Finally, another LSTM is applied to each station to aggregate both temporal and spatial relations for station i . The final hidden state $h_{i,t}$ is selected as the context vector c_i for station i which stores all the information of the encoding, and then this vector is passed to the decoder as the initial state to be decoded as shown below.

$$\hat{h}_{i,t} = LSTM(\hat{h}_{i,t-1}, g_{i,t}) \quad (15)$$

$$c_i = \hat{h}_{i,T'} \quad (16)$$

D. Decoder

In the decoder, a separate LSTM is used to decode context vector c_i to obtain the multistep outputs. The LSTM part is the same as the equations (2)-(7), while the initial state is set as

the context vector which stores all the information of previous time steps.

$$h_{i,0} = c_i, \quad (17)$$

$$h_{i,t} = LSTM(\hat{h}_{i,t-1}, x_{i,t}) \quad t > 0 \quad (18)$$

Furthermore, we employ the attention mechanism which is widely used in most of NLP scenarios [17]. At a high-level, an attention mechanism enables our neural network to focus on relevant parts of the inputs more than the irrelevant parts when performing a prediction task. For example, if the current time step is Sunday then the information of the Sunday before a week are considered as much help to predict the current output. Let t and t' denote the time step at decoder and encoder respectively, and the attention mechanism works as follows:

$$u_{i,t}^{t'} = q_a^T \tanh(W_a[h_{i,t} + \hat{h}_{i,t'}]) \quad (19)$$

$$a_{i,t}^{t'} = \frac{\exp(u_{i,t}^{t'})}{\sum_{t'=1}^{T'} \exp(u_{i,t}^{t'})} \quad (20)$$

$$s_{i,t} = \sum_{t'=1}^{T'} a_{i,t}^{t'} \hat{h}_{i,t'} \quad (21)$$

Where the weight $a_{i,t}^{t'}$ measures the importance of the time step t' in t . Here $a_{i,t}^{t'}$ is derived by comparing the current hidden state $h_{i,t}$ with the previous spatial-temporal hidden state $\hat{h}_{i,t'}$. The attention vector $s_{i,t}$ is a weighted sum of hidden states in each previous time step t' .

Finally, we concatenate attention vector s_i with current hidden state h_i as h_i' . Then we feed h_i' to a fully connected layer and get the final prediction. Noted that in this work, we predict inflow and outflow simultaneously.

$$\tilde{h}_{i,t} = h_{i,t} \oplus s_{i,t} \quad (22)$$

$$[\hat{y}_{i,t}^{in}, \hat{y}_{i,t}^{out}] = W_y \tilde{h}_{i,t} + b_y \quad (23)$$

Since this is a multistep problem, the loss function is defined as the mean squared error:

$$loss = \frac{1}{T} \sum_{t=1}^T (y_{i,t}^{in} - \hat{y}_{i,t}^{in})^2 + (y_{i,t}^{out} - \hat{y}_{i,t}^{out})^2 \quad (24)$$

IV. EXPERIMENT

A. Dataset

We use a large-scale car-sharing dataset provided by EV-CARD, which is one of the biggest hourly rental operators in China. The dataset contains the orders from 5/1/2017 to 7/31/2017 in Shanghai, and there are about 15,000 orders per day. The order includes pick-up station, drop-off station, pick-up time and drop-off time. Weather data is collected from JUHE¹ website. POI data is collected through AMap API²,

¹<https://www.juhe.cn/docs/api/id/277>

²<https://lbs.amap.com/>

which contains 15 primary categories. For each station, we collected POIs within 1km around it and represented them in a vector, whose entry is the number of a specific POI category. We summarize the statistics of the dataset in Table I.

The data from 5/1/2017 to 6/30/2017 are used for training (61 days), and the data from 7/1/2017 to 7/31/2017 are used for testing (31 days).

TABLE I
DATASET STATISTICS

Data Source		EVCARD	
Time from	5/1/17	7/1/17	
to	6/30/17	7/31/17	
#days	61	31	
#stations	1433	1433	
#orders	964,531	498,856	

Data Source		AMap API	
POI type	number	POI type	number
food	520,779	shopping	1,065,138
life service	550,200	sports	103,723
medical service	80,934	accommodation	56,490
tourist	11,773	residence	217,018
government	138,298	education	161,613
transportation	248,822	finance service	81,673
enterprises	455,720		

B. Experiment settings

In the experiment, we use the past 14-day historical data to predict the flow in the next 7 days. The number of hidden layers for LSTM is two with 64 hidden units. The dimension of graph convolution is set to 64. Adam [18] is selected as the optimization algorithm, and the initial learning rate is set to 0.001. Here 10% of the training data were selected as the validation set for parameter tuning, and early stopping is used. To speed up convergence, teacher forcing [19] is applied, which means we feed the actual flow to the decoder at the training stage. We perform Xavier initialization to initialize all the trainable parameters. The training process takes about 8 hours on a single TITAN XP GPU.

C. Baseline & Metric

We compare the proposed model (AMGC-Seq2Seq) with the following methods:

- **Historical Average (HA):** The historical average model predicts the flow by using the average value of history. In our experiment, the prediction is the average from the same time in previous weeks.
- **Autoregressive Integrated Moving Average (ARIMA):** ARIMA is a widely used time series prediction model. There are three parameters (p, d, q) need to be set for the model. The degree of differencing is set as $d = 1$. Here p and q are determined by grid search on the training set.
- **Seq2Seq:** Sequence to sequence model has been proved to be effective for time series prediction problem. Same as our model, Seq2Seq model is trained on all stations.
- **AGC-Seq2Seq [15]:** AGC-Seq2Seq is a graph based seq2seq model to predict traffic speeds. The encoder of

that model is different from ours in two aspects: (1) They employ only one graph while we combine multi-graphs, and (2) we have one more LSTM which is used to capture the local temporal relations for each station.

We use Mean Average Percentage Error (MAPE) and Rooted Mean Square Error (RMSE) to evaluate the proposed model, which are defined as follows:

$$MAPE = \frac{1}{NT} \sum_{i=1}^N \sum_{j=1}^T \frac{|\hat{y}_{t+j}^i - y_{t+j}^i|}{y_{t+j}^i} \quad (25)$$

$$RMSE = \sqrt{\frac{1}{NT} \sum_{i=1}^N \sum_{j=1}^T (\hat{y}_{t+j}^i - y_{t+j}^i)^2} \quad (26)$$

D. Performance comparison

Table II shows the performance of the proposed model compared to all other competing models. AMGC-Seq2Seq achieves the lowest RMSE (6.15) and the lowest MAPE (23.66) among all the methods. More specifically, HA and ARIMA perform poorly, as they rely on only historical data for prediction. Deep learning methods, including Seq2Seq, AGC-Seq2Seq, and AMGC-Seq2Seq, which are able to model the spatial-temporal relations, generally outperform the traditional methods. Compared with AGC-Seq2Seq, the proposed model further utilizes multi-graph convolution and one more LSTM to capture the local temporal relation for the individual station, which results the lowest RMSE and MAPE.

TABLE II
PERFORMANCE OF DIFFERENT METHODS

Method	RMSE	MAPE(%)
HA	8.03	30.23
ARIMA	7.54	27.89
Seq2Seq	6.73	25.93
AGC-Seq2Seq	6.48	24.82
AMGC-Seq2Seq	6.15	23.66

E. Effect of multi-graph convolution

Here we study the effect of multi-graph convolution. Table III shows the results when we only use a single graph (distance or POI graph) for prediction. According to the results, we observe that a single graph can be worse than the baseline method(e.g., the model of POI graph yields a result which is worse than Seq2Seq model). However, by combining them, our model beats the AGC-Seq2Seq with the lowest RMSE and MAPE, which proves the effectiveness of the proposed multi-graph convolution.

TABLE III
PERFORMANCE OF DIFFERENT GRAPH

Method	RMSE	MAPE
AGC-Seq2Seq + POI Graph	7.46	30.26
AGC-Seq2Seq + Distance Graph	6.52	25.35
AMGC-Seq2Seq	6.15	23.66

F. Effect of attention

Figure 2 shows the prediction of station 1 from 7/6 to 7/26, where the data from 7/6 to 7/19 are used as the historical data, and the flow from 7/20 to 7/26 are predicted by our model. In the attention mechanism, $a_{i,t}^t$ in eq.(20) measures the relevance of the historical information in the predicted state. The corresponding attention heatmap of station 1 is depicted in Figure 3, where the darker the color is, the more the relevance there are between two dates. One interesting finding is that when predicting flows on weekdays, the model tends to look at the latest history. However, when predicting flows on weekends (7/22 and 7/23 in this example), the model tends to look at the information on weekends from the history (7/15, 7/16, 7/8 and 7/9 in this example). These results confirm that our model can automatically capture the relevant information to make a more robust prediction.

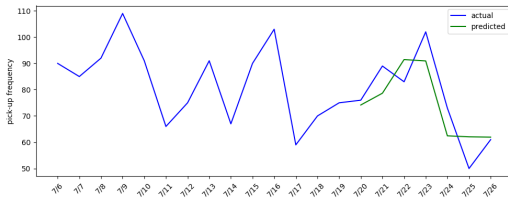


Fig. 2. Outflow of station 1

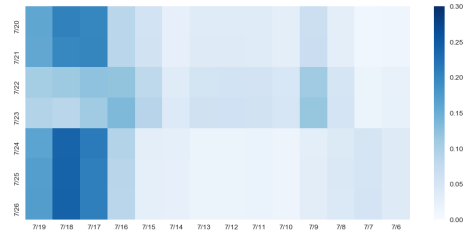


Fig. 3. Attention heatmap of station 1

V. CONCLUSION

In this paper, we propose a novel deep learning model AMGC-Seq2Seq for flow prediction in a car-sharing system. There are two novelties of the proposed model. The first is that we utilize multi-graph convolution to model the spatial relations from different aspects. The second is that we incorporate LSTMs with a graph convolution network in a "hamburger" structure which capture both spatial and temporal relations effectively. We evaluated the model on a large-scale real-world car-sharing dataset from EVCARD. The experiment results show that the proposed model achieved better results than state-of-the-art baselines. In future, we plan to investigate the following aspects: (1) evaluate the proposed model on other datasets. (2) incorporate more diverse features (e.g., road network) in a car-sharing system.

ACKNOWLEDGMENT

This research is supported by the National Key R&D Program of China under Grant (No.2018YFB0505000,

2018YFB0505000) and the Shanghai Committee of Science and Technology under Grant (No.17511107303, 17511110202). This research is also supported by EVCARD.

REFERENCES

- [1] Richard Katzew. Car sharing: A new approach to urban transportation problems. *Analyses of Social Issues and Public Policy*, 3(1):65–86, 2003.
- [2] Angela Febbraro, Nicola Sacco, and Mahnam Saeednia. One-way carsharing: solving the relocation problem. *Transportation Research Record: Journal of the Transportation Research Board*, (2319):113–120, 2012.
- [3] Diana Jorge and Gonalo Correia. Carsharing systems demand estimation and defined operations: a literature review. *European Journal of Transport and Infrastructure Research*, 13(3), 2013.
- [4] Billy M Williams and Lester A Hoel. Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results. *Journal of transportation engineering*, 129(6):664–672, 2003.
- [5] Bo Zhou, Dan He, Zhili Sun, and Wee Hock Ng. Network traffic modeling and prediction with arima/garch. In *Proc. of HET-NETs Conference*, pages 1–10, 2005.
- [6] Shashank Shekhar and Billy M Williams. Adaptive seasonal time series models for forecasting short-term traffic flow. *Transportation Research Record*, 2024(1):116–125, 2007.
- [7] Yexin Li, Yu Zheng, Huichu Zhang, and Lei Chen. Traffic prediction in a bike-sharing system. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 33, 2015.
- [8] Dingxiong Deng, Cyrus Shahabi, Ugur Demiryurek, Linhong Zhu, Rose Yu, and Yan Liu. Latent space model for road networks to predict time-varying traffic. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1525–1534, 2016.
- [9] Yongxin Tong, Yuqiang Chen, Zimu Zhou, Lei Chen, Jie Wang, Qiang Yang, Jieping Ye, and Weifeng Lv. The simpler the better: a unified approach to predicting original taxi demands based on large-scale online platforms. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1653–1662, 2017.
- [10] Zheng Zhao, Weihai Chen, Xingming Wu, Peter CY Chen, and Jingmeng Liu. Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.
- [11] Dong Wang, Wei Cao, Jian Li, and Jieping Ye. Deepds: supply-demand prediction for online car-hailing services using deep neural networks. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 243–254, 2017.
- [12] Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, Jieping Ye, and Zhenhui Li. Deep multi-view spatial-temporal network for taxi demand prediction. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [13] Di Chai, Leye Wang, and Qiang Yang. Bike flow prediction with multi-graph convolutional networks. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 397–400, 2018.
- [14] Pinlong Cai, Yunpeng Wang, Guangquan Lu, Peng Chen, Chuan Ding, and Jianping Sun. A spatiotemporal correlative k-nearest neighbor model for short-term traffic multistep forecasting. *Transportation Research Part C: Emerging Technologies*, 62:21–34, 2016.
- [15] Zhengchao Zhang, Meng Li, Xi Lin, Yinhai Wang, and Fang He. Multistep speed prediction on traffic networks: A graph convolutional sequence-to-sequence learning approach with attention mechanism. *arXiv preprint arXiv:1810.10237*, 2018.
- [16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [17] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
- [18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [19] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Chinese Text Relation Extraction with Multi-instance Multi-label BLSTM Neural Networks

1st Liubo Ouyang

Department of Software Engineering
Hunan University
Changsha, China
oylb@hnu.edu.cn

2nd Hui Tang

Department of Software Engineering
Hunan University
Changsha, China
thlhl2525@hnu.edu.cn

3rd Guangyi Xiao

Department of Software Engineering
Hunan University
Changsha, China
gyxiao@hnu.edu.cn

Abstract—Recently, deep learning models have emerged as powerful tools for relation extraction. However, little work has been done on relation extraction for the Chinese language. One major challenge for relation extraction in Chinese texts is that Chinese sentences have no obvious word segmentation. This ambiguity increases the possibility of word segmentation errors. Another challenge is the lack of broad-scale Chinese text datasets. In this paper, we propose an attention-based multi-instance multi-label bidirectional long short-term memory network for distantly supervised Chinese relation extraction. Our model takes Chinese character embeddings and position embeddings as input without Chinese word segmentation errors. Then, the attention mechanism is used to extract richer Chinese character and sentence features. Finally, we handle the multi-label nature of relation extraction by using multi-label loss functions in the neural network classifier. Based on the idea of distant supervision, we constructed a new dataset for relation extraction in Chinese texts. Experiments on this dataset show that our method has achieved relatively high performance, and that the proposed network architecture is suitable for Chinese relation extraction. Furthermore, we also ran experiments on a popular English benchmark dataset, and the results show that our method is superior to some existing methods.

Index Terms—Chinese relation extraction, distant supervision, attention, bidirectional long short-term memory network (BLSTM).

I. INTRODUCTION

Relation extraction is the detection and identification of semantic relations between natural language text entities [1]. The relation extraction problem can be formally described as follows. Given a sentence s and two entities, e_1 and e_2 , in s , predict the relationship type r of e_1 and e_2 in the sentence s . The candidate set of r is a predefined relation set R , and the output is usually a triple (e_1, e_2, r) [2]. As one of the key tasks of natural language processing (NLP), relation extraction has high significance for many applications of NLP, such as question answering and knowledge graphs [3].

Distant supervision is proposed to automatically generate labeled training data by aligning knowledge bases and text for relation extraction [4]. The main idea of distant supervision is that if two entities have a relationship in the knowledge base, then all sentences containing the two entities will represent this relationship. Distant supervision solves

the problem of constructing labeled training data and saves human resources needed for manually labeling data. However, since a sentence containing two entities does not necessarily represent a corresponding relationship, a large amount of noise data is inevitably introduced. Therefore, distant supervision is modeled as a multi-instance multi-label classification problem [5]- [7]. In recent years, deep learning models have been applied in relation extraction [8]. The most commonly used models for relation extraction tasks include recursive neural networks (RecNN) [9], convolutional neural networks (CNN) [10], recurrent neural networks (RNN) [11]. Moreover, LSTM network is an improved variant of RNN that has been widely applied to natural language processing tasks and has achieved good performance [12]. Adding an attention mechanism to the model and weighting the data sequence can effectively improve sequence learning and boost the system performance [13] [14].

The research on relation extraction for Chinese texts is relatively less common than that for English ones [15]. This scarcity can be ascribed to three main difficulties of Chinese relation extraction. Firstly, there is no obvious separation between Chinese words; Chinese words are composed of characters whose combinations are highly complex and ambiguous. Secondly, current Chinese word segmentation systems still have considerable errors that introduce noise into the relation extraction task. Finally, there is a lack of a Chinese corpus for relation extraction.

In this paper, we propose an attention-based multi-instance multi-label bidirectional long short-term memory network (ATT+MIML+BLSTM) for distantly supervised Chinese texts relation extraction. It is sufficient to use raw Chinese sentences as input. The BLSTM and character-level attention modules are used to obtain the important semantic information in each sentence, and then the final sentence vector representation is obtained through sentence-level attention. Also, the multi-label loss function is used in the network model to deal with overlapping relations. In view of the lack of a Chinese dataset, and based on the distant supervision concept, we used CN-DBpedia of Fudan Knowledge Factory ¹ to obtain the identified entity pairs, and aligned them in the SogouCS2012

news text corpus ² to construct a large Chinese character relationship dataset. The model proposed in this paper is evaluated on this constructed dataset as well as on an English benchmark dataset. The results show that the proposed model is suitable for relation extraction in Chinese texts and achieves good performance.

The contributions of this paper are summarized as follows:

- Proposing a deep learning model for distantly supervised relation extraction in Chinese texts.
- Using Chinese character vectors as model inputs to avoid introducing Chinese word segmentation noise into the relation extraction process.
- Making full use of the information of each word in the sentence and all the sentence information with the same entity pair, and handling the overlapping relationships of entity pairs.
- Performing extensive experiments on Chinese and English benchmark datasets showing the suitability and good performance of our model for relation extraction in both languages.

II. RELATE WORK

Over the years, many methods for relation extraction have been proposed. These methods can be mainly divided into three categories: supervised, semi-supervised and unsupervised approaches. Supervised approaches treat relation extraction as a classification task and typically exhibit better performance in comparison to other approaches. However, supervised methods are very time consuming and labor intensive since they require a lot of labeled data. Distant supervision can solve this problem through automatic data labeling. However, distant supervision can cause false labeling problems. Riedel et al. [5] models distant supervision as a multi-instance single-label problem and selects for each entity pair. Furthermore, Hoffmann et al. [6] and Surdeanu et al. [7] noted that there might be multiple relations between entities and hence they cast relation extraction as a multi-instance multi-label learning problem.

Recently, with the emergence of deep learning, many scholars have begun to use deep learning to automatically learn features. In NLP, deep learning methods are mainly based on learning the distributed representation of each word, which is also called a word embedding [16]. Zeng et al. [17] proposed a convolutional neural network for relational classification. Furthermore, Zeng et al. [18] combined at least one multi-instance learning scheme with a neural network model to extract relationships using distant supervision. Zhou et al. [19] used a neural attention mechanism combined with a bidirectional long-term memory network (BLSTM) to obtain important semantic information in sentences. Lin et al. [14] showed that the establishment of a sentence-level attention mechanism for dynamically calculating the weights of multiple instances can be very effective in distantly supervised data. Jiang et al. [20] used the maximum pool across sentences

to select features in different sentences, then aggregated the most important features into the vector representation of each entity pair. In addition, they addressed relation extraction as a multi-label problem. Jat et al. [21] combined multiple complementary models to improve relational extraction, and introduced a new distantly supervised dataset that eliminated the test data noise present in all previous benchmark datasets. Feng et al. [22] applied reinforcement learning to distantly supervised relationship extraction. Relevant side information from KB has been utilized for relation extraction [23] [24].

In the study of Chinese texts relation extraction, lexical and syntactic features are usually used to extract feature vectors, and the classifier SVM can be used for Chinese texts relation extraction [25]. The performance of Chinese relation extraction has been improved by clustering and pattern matching the feature vectors extracted by dependence relationship and parts of speech labeled in relational schema [26]. In the works of [27] and [28], all can reveal that performance of Chinese relation extraction based on kernel function has obtained significant improvement. However, The need for amounts of manually annotated corpora hinders the application of deep learning methods in Chinese texts relation extraction.

At present, relation extraction research and experiments are mainly focused on English. This paper focuses on distant supervision for relation extraction in Chinese texts. Based on the advantages of previous models, this paper proposes a multi-instance multi-label BLSTM model equipped with an attention mechanism. This model can obtain the rich semantic information of the Chinese sentences and handle overlapping relations.

III. MODEL

As shown in Fig.1, the proposed ATT+MIML+BLSTM model consists of three main parts:

- **Sentence Representation:** Enter raw characters of the input sentence into the BLSTM model, and use the character-level attention mechanism to weight the output of each time step of the BLSTM network. Merge the character-level features of each time step into a sentence-level feature vector.
- **Entity-pair Representation:** Use sentence-level attention to give different weights to different statements, implicitly discarding some noise statements.
- **Multi-label Classification:** Multi-label classification is performed by using multiple binary classifiers corresponding to a class of relationships.

A. Sentence Representation

Sentence-level features are designed to construct a distributed representation for each sentence. As shown in Fig.2, we first convert the Chinese characters in the sentence into real-valued vectors. The higher features of the sentence are then extracted by BLSTM. The final character-level attention weights the output of each time step of the BLSTM. The character-level features of each time step are then merged into a sentence-level feature vector.

²<http://www.sogou.com/labs/>

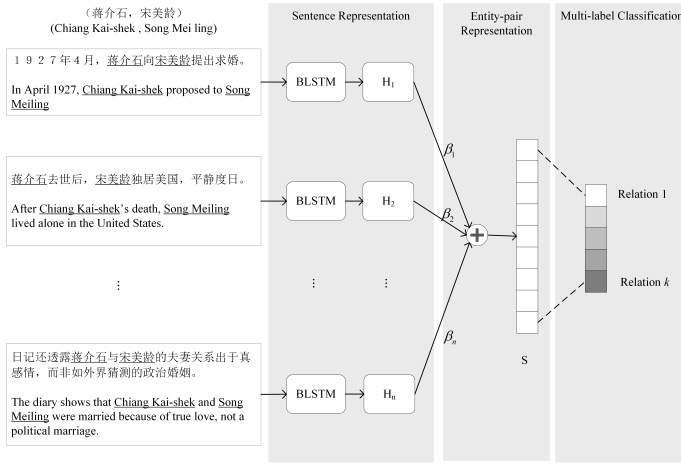


Fig. 1. The architecture of ATT+MIML+BLSTM used for distant supervised relation extraction.

Input Representation Layer. The input for each sentence consists of two embeddings:

Character Embedding: Chinese words are composed of characters, the combinations of which are very complicated. As well, there is no obvious separation between Chinese words. Chen et al. [29] observed experimentally that character features are more suitable for Chinese relation extraction tasks than word features. Based on this observation, we use character embeddings to avoid introducing word segmentation errors into the relation extraction process. The model takes as an input each raw Chinese character in a Chinese sentence s . Therefore, given a sentence $s = \{x_1, x_2, \dots, x_m\}$ consisting of m Chinese characters, we convert each Chinese character into a real-value vector by looking up the pre-trained embedded matrix $V \in \mathbb{R}^{d_w \times |V|}$ (where d_w is the dimension of the character embedding and $|V|$ is vocabulary size).

Position Embedding: Position embeddings are used to represent the structural information of sentences, and two different dimensional position vectors are constructed by the relative distances between each Chinese character and the two entities $e1$ and $e2$.

We concatenate all the character embeddings and position embeddings to get a sequence of vectors $w = \{w_1, w_2, \dots, w_m\}$ (where $w_i \in \mathbb{R}^d$, $d = d_w + 2 \times d_p$ and d_p is the dimension of the position embedding) and set such a sequence as the model input.

BLSTM Layer. The LSTM model can effectively alleviate the long-distance dependence problem of RNN and CNN, and it has been improved and promoted recently by Graves [30]. In many problems, LSTM has achieved considerable success and has been widely used.

In LSTM, the forget gate f_t determines how much information is discarded in the cell state. The gate has three inputs: x_t is the input of the current time step, h_{t-1} is the output of the previous LSTM cell, and c_{t-1} is the memory of the previous cell.

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (1)$$

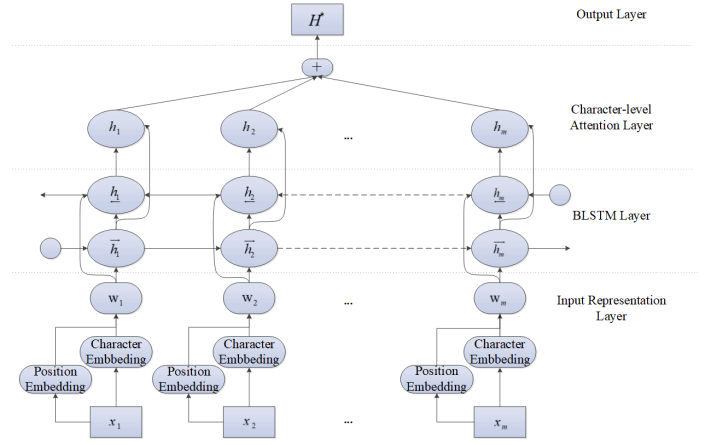


Fig. 2. Detailed structure of the sentence representation.

where, W_{xf} , W_{hf} , W_{cf} , and b_f are weight matrices.

The input gate i_t determines what information will be updated and has the same input as the forget gate f_t :

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (2)$$

where, W_{xi} , W_{hi} , W_{ci} , and b_i are weight matrices.

Then create a new candidate value vector \tilde{c}_t that will be added to the state:

$$\tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + W_{cc}c_{t-1} + b_c) \quad (3)$$

where, W_{xc} , W_{hc} , W_{cc} , and b_c are weight matrices.

Update current cell status c_t :

$$c_t = i_t \tilde{c}_t + f_t c_{t-1} \quad (4)$$

Finally, how much information is controlled by output gate o_t should be entered into the next LSTM cell.

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \quad (5)$$

$$h_t = o_t \tanh(c_t) \quad (6)$$

where, W_{xo} , W_{ho} , W_{co} , and b_o are weight matrices.

As can be seen from Figure 2, the BLSTM layer contains forward and reverse LSTM networks. The output for the i^{th} Chinese character from the BLSTM layer is combined by the forward and reverse outputs by element-wise summation:

$$h_i = [\vec{h}_i \oplus \overleftarrow{h}_i] \quad (7)$$

Character-level Attention Layer. Following the approach of [20], we use character-level attention to capture important information in sentences and improve the accuracy of sentence representation. The vector sequence $H = \{h_1, h_2, \dots, h_m\}$ output by the BLSTM layer is weighted to obtain the final representation of the sentence.

First apply a non-linear activation function on h_i :

$$N_i = \tanh(h_i) \quad (8)$$

Then calculate the weight of the Chinese characters in the sentence:

$$\alpha_i = \text{softmax}(W^T N_i) \quad (9)$$

where, W^T is the training parameter.

The vector representation r of the sentence is obtained by weighting the N_i :

$$r = \sum_{i=1}^m \alpha_i N_i \quad (10)$$

Finally, a non-linear activation function is applied to r to get the final representation of the sentence H^* .

$$H^* = \tanh(r) \quad (11)$$

B. Entity-pair Representation

Distant supervision produces a lot of noise or mislabeled data, and direct use of supervised methods to classify relationships is very ineffective. In order to solve the problem of mislabeling, this paper proposes the construction of sentence-level relational attention on multiple instances by using the selective attention mechanism to weigh sentence vectors and weaken the weight of noisy instances dynamically [14]. Suppose there is a set S containing n sentences of the same entity pair, and the set S is represented as having a real-valued vector, i.e. $S = \{H_1^*, H_2^*, \dots, H_n^*\}$, where H_i^* is the representation of the sentence obtained in the subsection A.

First, calculate the degree to which a sentence H_i^* matches the corresponding relationship.

$$D_i = H_i^* \cdot A \cdot l \quad (12)$$

where A is a weighted diagonal matrix and l is the vector representing the relationship. Therefore, the size of D_i depends on the size of the mapping of H_i^* on l , and sentences that are more closely related to the entity relationship can achieve larger values. Then, we can get the weight β_i :

$$\beta_i = \text{softmax}(D_i) \quad (13)$$

Then for the sentence set S , it can be calculated as the weighted sum of all the sentences in the set:

$$S = \sum_i \beta_i H_i^* \quad (14)$$

C. Multi-label Classification

In this paper, we formalize distant supervision as a multi-instance multi-label learning problem. In this section, we will handle the overlapping relationship of entity pairs. The sentence set vector S obtained in the subsection B, and then get o through a layer of network:

$$o = MS + b \quad (15)$$

where M is the weight matrix of all relational vectors and b is a bias. Thus o represents the confidence score for each relation label.

Then, we use multiple binary functions to do multi-label classification. In particular, we calculate the probability of each relationship. The relationship label is considered accurate if the relationship probability exceeds a certain threshold.

$$p_i = \text{sigmoid}(o_i), i = \{1, 2, \dots, k\} \quad (16)$$

where k is the number of relation labels.

We set the binary label vector y to represent the set of true relationships between pairs of entities, where 1 represents a relation in the set and 0 otherwise. Finally, we use the cross entropy of the sigmoid function as the loss function:

$$\text{loss} = - \sum_{i=1}^k y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \quad (17)$$

where $y_i \in \{0, 1\}$ is the true value on label i .

We train the model in an end-to-end manner. We use Adam to optimize the loss function [31]. In the training phase, we used a dropout mechanism in the BLSTM layer to prevent overfitting [32]. For the testing phase, our method selects a relation with a probability of more than 0.5 as a predicted label.

IV. EXPERIMENTS

A. Experiments on Chinese Character Relationship Dataset

Building Chinese datasets. Using distant supervision, we can automatically construct a Chinese dataset, which avoids labor intensive of manually labeling data. The details of these steps are as follows.

Step 1: Obtain entity pairs with a defined relationship. The large-scale structured encyclopedia CN-DBpedia, developed and maintained by the Fudan University Knowledge Factory Laboratory, is the earliest and the largest open Chinese knowledge extraction system. We set up a list of seed names, use the free API provided by the Fudan Knowledge Factory to obtain the corresponding personal relationships, and then add the character entities not included in the entity list to the list, and iterate over and over again.

Step 2: Align the entity pairs with certain relationships with the Chinese text corpus. The Chinese text corpus uses SogouC-S from Sogou Lab, which is one of the most comprehensive Chinese text corpus resources. Pure text is obtained by data preprocessing through the corpus. The text corpus is aligned with the entity pairs of the first step to get the statements containing the entity pairs.

Finally, the dataset contains eight kinds of relationships (cooperation, friends, couples, parenthood, lovers, faculty-student, brother and sister, others). We divide the training set and the test set according to a ratio of 9:1. The training set contains 220,160 sentences and the test set contains 24,463 sentences.

Experimental settings and evaluation metrics. Chinese character embeddings are trained on the Chinese Wikipedia corpus by the word2vec tool³. We set the dimension of character embedding to 100 and the dimension of position embedding to 5. Position embeddings are randomly initialized with uniform distribution between $[-1, 1]$. At BLSTM layer,

³<https://code.google.com/p/word2vec/>

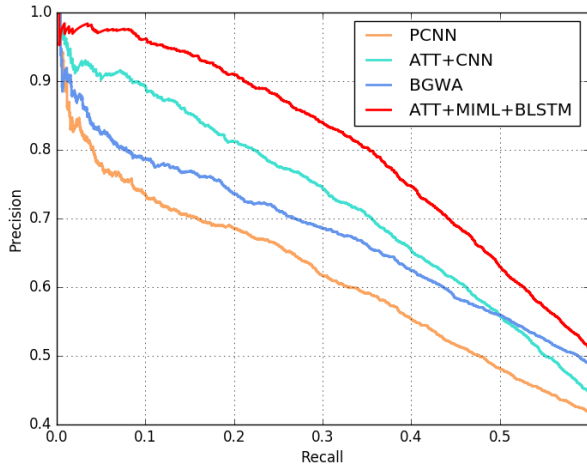


Fig. 3. Performance comparison of our method with three baselines for Chinese dataset.

TABLE I
COMPARISON OF P@N RESULTS BETWEEN OUR MODEL AND OTHER MODELS.

	PCNN	BGWA	ATT+CNN	ATT+MIML+BLSTM
P@100	0.87	0.91	0.95	0.97
P@200	0.845	0.885	0.93	0.955
P@300	0.817	0.85	0.923	0.946
Mean	0.844	0.882	0.934	0.957

the number of LSTM hidden units is set to 230. Meanwhile, we use a batch of 64 entity pairs, set learning rate to 0.001 and dropout keep probability to 0.5.

Following previous work [14] [18], We compare the performance of each model with the aggregate curves Precision/Recall(PR) curves and Precision@N(P@N).

We choose the following three deep learning models as baselines: (1) **PCNN**: A piecewise max-pooling over CNN based relation extraction model. (2) **ATT+CNN**: A CNN based model with sentence-level attention. (3) **BGWA**: A piecewise max-pooling over bidirectional gated recurrent unit based model with word-level attention. They are all superior to traditional methods and are also significant works for relation extraction.

Experimental results. The curves in Fig.3 show that our model has a relatively high accuracy and recall rate compared to other models when we perform Chinese relation extraction. In other words, under the same recall rate, the accuracy of Chinese relation extraction using ATT+MIML+BLSTM is higher than other models. We observed that the PR curve of the neural network method (ATT+CNN, BGWA, ATT+MIML+BLSTM) that introduces the attention mechanism is significantly higher than the ordinary neural network method PCNN. It can be seen that the attention mechanism can improve the performance of the model. In addition, our model is superior to the CNN model which also adopts sentence-level attention, which shows that BLSTM combined with word attention can obtain richer semantic information, and considering multi-label problem can

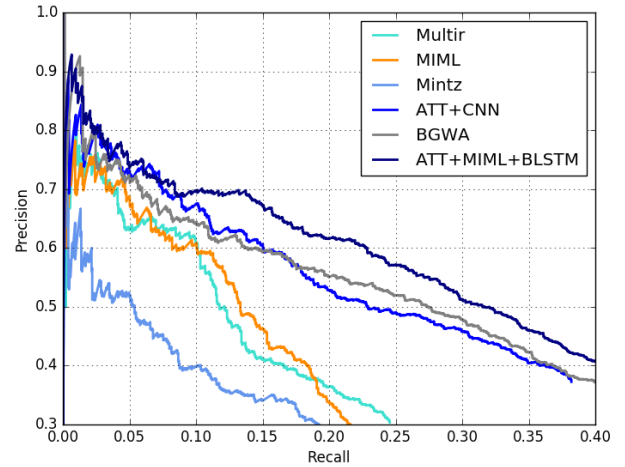


Fig. 4. Performance comparison of our method with five popular methods for English benchmark dataset.

TABLE II
STATISTICS OF THE NYT10 DATASET.

	Sentences	Entity Pairs	Relations
Training	522611	281270	53
Testing	172448	96678	53

improve model performance. Table I shows the results using the P @ N metric. Similar to Figure 4, our approach is superior to other methods overall. And when N is smaller, our method accuracy is marginally higher than others.

B. Experiments on the English Benchmark Dataset

In order to evaluate the performance of our model on other languages, we conducted a comparative experiment on the English dataset proposed by Riedel in 2010 [5]. The dataset is generated by the heuristic matching of Freebase and the New York Times(NYT), and is a popular benchmark dataset. As shown in Table II, the dataset contains 53 relationships (including "NA", indicating that there is no relationship between entity pairs). We used sentences from 2005 to 2006 as training data and sentences from 2007 as test data.

The parameter settings for the English experiments are like those of the Chinese ones, except that the input is word embedding, the word embedding dimension is 50, and the batch size is 50.

We compare our model with three traditional methods(Mintz [4], Multir [6] and MIML [7]) and two popular neural-based methods(ATT+CNN and BGWA). They are major works for English relation extraction based on distant supervision.

Fig.4 clearly shows that the PR curve of our model (ATT+MIML+BLSTM) is above those of the other models, so our model is not only superior to traditional methods but also neural-based methods. Since the size of the Chinese dataset we constructed is smaller than the English benchmark dataset and the amount of noise data is different, there is a small error in the experimental results on the two datasets. However, we

can still conclude that our model is superior to other models, especially for Chinese relation extraction.

V. CONCLUSION

In this paper, we propose a multi-instance multi-label neural network model based on the attention mechanism for Chinese relation extraction. It can not only obtain the rich semantic information of the sentence, but also consider the multi-relationship problem of the entity pair. Experiments on a Chinese dataset based on distant supervision prove that the neural network model combining attention mechanism and multi-label learning can achieve good results, and the proposed method has better performance than most existing methods on the benchmark English dataset. In the future, we hope to build a large-scale and standardized Chinese corpus, and further study how different loss functions affect the performance of the model.

REFERENCES

- [1] N. Bach and S. Badaskar, "A review of relation extraction," *Literature review for Language and Statistics II*, vol. 2, 2007.
- [2] I. Hendrickx, N. K. Su, Z. Kozareva, P. Nakov, M. Pennacchiotti, L. Romano, and S. Szpakowicz, "Semeval-2010 task 8: multi-way classification of semantic relations between pairs of nominals," in *Workshop on Semantic Evaluations: Recent Achievements & Future Directions*, 2009.
- [3] Q. Zhang, M. Chen, and L. Liu, "A review on entity relation extraction," in *2017 Second International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, vol. 00, Dec. 2018, pp. 178–183. [Online].
- [4] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*. Association for Computational Linguistics, 2009, pp. 1003–1011.
- [5] S. Riedel, L. Yao, and A. McCallum, "Modeling relations and their mentions without labeled text," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 148–163.
- [6] R. Hoffmann, C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld, "Knowledge-based weak supervision for information extraction of overlapping relations," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 541–550.
- [7] M. Surdeanu, J. Tibshirani, R. Nallapati, and C. D. Manning, "Multi-instance multi-label learning for relation extraction," in *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*. Association for Computational Linguistics, 2012, pp. 455–465.
- [8] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [9] C. Goller and A. Kuchler, "Learning task-dependent distributed representations by backpropagation through structure," *Neural Networks*, vol. 1, pp. 347–352, 1996.
- [10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [11] J. L. Elman, "Distributed representations, simple recurrent networks, and grammatical structure," *Machine learning*, vol. 7, no. 2-3, pp. 195–225, 1991.
- [12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.
- [14] Y. Lin, S. Shen, Z. Liu, H. Luan, and M. Sun, "Neural relation extraction with selective attention over instances," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2016, pp. 2124–2133.
- [15] W. Zirui, M. Fang, and J. Libiao, "Review of chinese entity relation extraction," in *2017 3rd IEEE International Conference on Control Science and Systems Engineering (ICCSSE)*, Aug 2017, pp. 633–637.
- [16] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
- [17] D. Zeng, K. Liu, S. Lai, G. Zhou, and J. Zhao, "Relation classification via convolutional deep neural network," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, 2014, pp. 2335–2344.
- [18] D. Zeng, K. Liu, Y. Chen, and J. Zhao, "Distant supervision for relation extraction via piecewise convolutional neural networks," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1753–1762.
- [19] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, vol. 2, 2016, pp. 207–212.
- [20] X. Jiang, Q. Wang, P. Li, and B. Wang, "Relation extraction with multi-instance multi-label convolutional neural networks," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, 2016, pp. 1471–1480.
- [21] S. Jat, S. Khandelwal, and P. Talukdar, "Improving distantly supervised relation extraction using word and entity based attention," 2018.
- [22] J. Feng, M. Huang, L. Zhao, Y. Yang, and X. Zhu, "Reinforcement learning for relation classification from noisy data," *CoRR*, vol. abs/1808.08013, 2018.
- [23] G. Ji, K. Liu, S. He, and J. Zhao, "Distant supervision for relation extraction with sentence-level attention and entity descriptions," pp. 3060–3066, 2017.
- [24] S. Vashishth, R. Joshi, S. S. Prayaga, C. Bhattacharyya, and P. Talukdar, "Reside: Improving distantly-supervised neural relation extraction using side information," 2018.
- [25] W. Che, T. Liu, and S. Li, "Automatic entity relation extraction," *Journal of Software*, vol. 19, no. 2, pp. 2–7, 2005.
- [26] L. Yu and K. Zhou, "A dynamic local path planning method for outdoor robot based on characteristics extraction of laser rangefinder and extended support vector machine," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 30, no. 02, p. 1659004, 2016.
- [27] H. Zhang, S. Hou, and X. Xia, "A novel convolution kernel model for chinese relation extraction based on semantic feature and instances partition," in *Computational Intelligence and Design (ISCID), 2012 Fifth International Symposium on*, vol. 1. IEEE, 2012, pp. 411–414.
- [28] A. Yang, Y. Du, and Q. Meng, "Extracting personae interactive relation in chinese microblog based on an improved dependency trigram kernel," *DEStech Transactions on Engineering and Technology Research*, no. ICMITE2016, 2016.
- [29] Y. Chen, D. Zheng, and T. Zhao, "Chinese relation extraction based on deep belief nets," *Journal of Software*, vol. 23, no. 10, pp. 2572–2585, 2012.
- [30] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.
- [31] D. Kinga and J. B. Adam, "A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, vol. 5, 2015.
- [32] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

A Convolutional Neural Network Pruning Method Based On Attention Mechanism

XiaoJie Wang, WenBin Yao* and Huiyuan Fu

Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia

Beijing University of Posts and Telecommunications

Beijing, China

{then, yaowenbin, fhy}@bupt.edu.cn

Abstract—Pruning effectively reduces the size of neural networks, which facilitates deployment of neural networks in production environment, especially in embedded systems with limited computing resources. In this paper, we propose a convolutional neural network pruning method based on attention mechanism. We add an attention module to model to generate scaling factors for channels. The scaling factors are considered as channels' importance score, thus filters and convolution kernels corresponding to channels with lower importance score are removed. Our method has the ability to learn importance of channels during training, instead of considering only the direct impact of parameters like existing methods. Moreover, it does not depend on any dedicated libraries, so could be combined with other compression methods for better performance. In experiments, we prune about 90% parameters in VGGNet with 0.67% accuracy drop and prune about 50% parameters in ResNet-56 with 1.02% accuracy drop.

Index Terms—convolutional neural network; pruning; attention mechanism

I. INTRODUCTION

In recent years, with the rapid development of deep learning, convolutional neural networks have achieved excellent performance in many fields, such as computer vision, speech recognition and natural language processing, etc. However, these extraordinary performances are at the expense of high computational and storage demands. Thus neural network compression technique has great significance for deploying a deep convolutional neural network on embedded devices (like mobile phone and IOT device) with constrained resource.

Many works have been proposed to compress deep models, including network quantization [1], [2], matrix decomposition [3], [4], and knowledge distillation [5], [6]. Pruning is one of the most effective compression methods, it aims to remove redundant parameters in neural networks. Which parameters are redundant depends on importance measurement of parameters. Early studies measure the importance of parameters by calculating second derivative of parameters to loss function [7], [8]. In spite of their success, second derivative has expensive calculation and memory overhead. Thus, most recently proposed pruning methods are based on direct impact of parameters, such as parameter magnitude [9] or sparsity of

output [10], which do not take correlation between parameters and loss function into consideration.

In this paper, we propose a filter-level pruning method based on attention mechanism. In our method, an attention module (SEBlock [11]) is added to network to generate scaling factors for channels, then we consider scaling factors as channel importance score to guide filter-level pruning. The attention module adjusts scaling factors of channels according to loss function in backpropagation, which associates the importance score of parameters with loss function. By removing filters and convolution kernels corresponding to low importance score channels, we can effectively reduce size of deep model and maintain prediction accuracy.

We conduct experiments on CIFAR-10 dataset [12], results show that our method can remove about 90% parameters in VGGNet [13], with only 0.67% accuracy drop. Pruning on ResNet-56 [14] also be conducted to verify the effectiveness of our method on the network with shortcut connections. More serious decrease of accuracy appears on ResNet-56 since it is a compact network, but we still remove half of parameters with roughly 1% accuracy drop. Moreover, we compare our method with weight sum [9] and APoZ [10] on CIFAR-100 [12], the result shows that our method has better performance under the same pruning ratio.

II. RELATED WORK

LeCun et al. [8] proposed that some of the neurons connections in neural networks could be removed without decreasing model accuracy. Similar to LeCun's work, Hassibi et al. [7] used hessian matrix to get the second derivative of parameters. However, the spatial complexity of hessian matrix is $O(n^2)$ (n is the number of parameters), which has expensive memory cost on deep models. In order to avoid problem mentioned above, recent studies prefer to prune models according to direct impact of parameters. Han et al. [15] proposed that the magnitude of the parameters could reflect the importance of the parameters, they remove parameters below a certain threshold to get compact model. Deep compression [16] further compress deep neural networks with pruning, trained quantization and Huffman Coding. However, pruning neurons connections produces sparse matrices, which relies on specific operational libraries and hardware to exploit performance advantages [17].

Correspondence: yaowenbin@bupt.edu.cn

DOI reference number: 10.18293/SEKE2019-147

In order to avoid the limitation of pruning neurons connections, structural pruning method was proposed. Lebedev et al. [18] explored a structured sparsity learning method(SSL), which adds the regularization term of parameter group to loss function so that certain groups of parameters would shrink to zeros during training, eventually be removed safely. However, SSL still destroys the network structure and depends on dedicated libraries.

In recent researches [9], [10], [19], [20], filter-level pruning become an effective pruning method that completely avoid using dedicated libraries. Li et al. [9] calculated the sum of the absolute values(L1 paradigm) of weights in filters as their importance score. This method has strong limitations because L1 paradigm of filter does not reflect the feature extraction ability of filter. Hu et al. [10] observed the sparsity of the ReLU activation function and assumed that a neuron is unimportant if most outputs of the neuron with ReLU are zero. Although Hu's method considers more further effects of the filters, it still does not determine the importance of parameters according to loss function. The methods mentioned above are based on the direct impact the parameters, which does not well represent the effect of parameters on neural network's loss function. Meanwhile, they are artificially formulated, which leads to that human intervention is added in training process which violates the rules of end-to-end training in deep learning.

III. OUR METHOD

In this section, we would first introduce filter-level pruning, which determines the granularity of our pruning method. Then, the attention module used in our method would be described. Next, average scaling factor formula would be presented. Finally, we would show overall steps of our method and pruning strategy adjustment on special model.

A. Filter-level pruning

As shown in Fig. 1, each layer of neural network consists of several filters, one filter has a set of convolution kernels. Instead of generating sparse matrices or destroying network structures, filter-level pruning removes entire filter, which maintains the regularity of the network. For example, channel2 would disappear when the filter2 marked by dotted lines are removed. In this case, convolution kernels process channel2 in next layer could be removed as well.

In a convolution layer, if we suppose its original parameter matrix is expressed as $W_{<I,W,H,C>}$, where I is the number of input channels, W is the width of convolution kernels, H is the height of convolution kernels and C is the number of filters. We can calculate the number of parameters in this layer as:

$$|W_{<I,W,H,C>}| = I \times W \times H \times C \quad (1)$$

If we set the filter pruning ratio of this layer to q , $q \times C$ filters in this layer would be removed, so that the number of remaining parameters in this layer can be calculated as:

$$|W_{<I,W,H,(1-q)C>}| = I \times W \times H \times (1-q)C \quad (2)$$

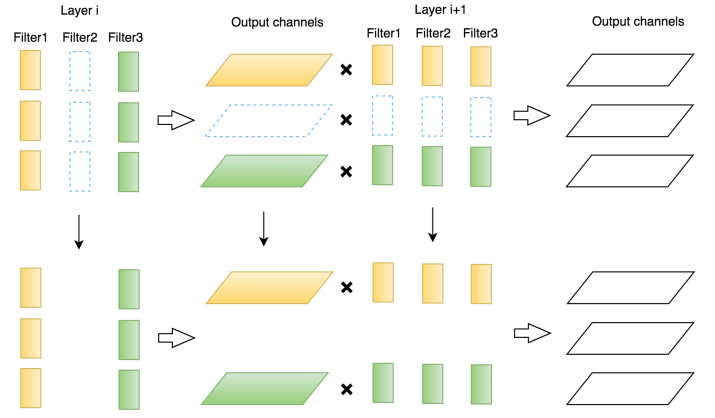


Fig. 1. Filter-level pruning.

If we set the filter pruning ratio of the front layer to p , $p \times I$ convolution kernels in this layer would be removed, so that the number of remaining parameters in this layer can be calculated as:

$$|W_{<(1-p)I,W,H,(1-q)C>}| = (1-p)I \times W \times H \times (1-q)C \quad (3)$$

Obviously, after pruning all layers, the total parameters of the convolution layer would be reduced to $(1-p) \times (1-q)$ of the original layer. In other words, $q + (1-q) \times p$ of the parameters would be removed.

B. Attention module

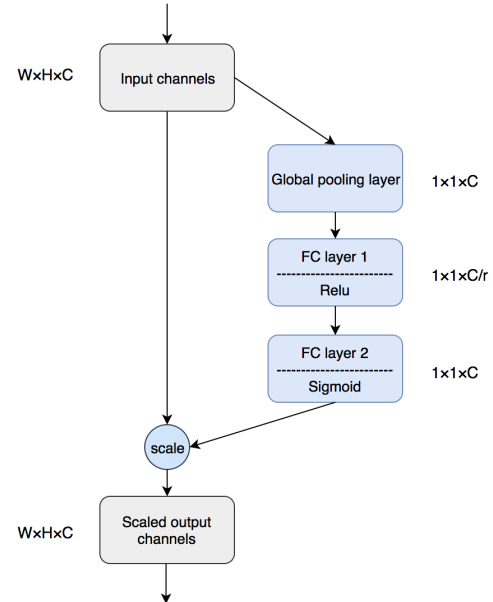


Fig. 2. SEBlock structure, the blue part is SEBlock.

As mentioned in related work, the filter-level pruning methods lack the consideration of correlation between parameters and loss functions due to computational complexity. To calculate importance score of filters according to loss function in a feasible computational complexity, we use attention module

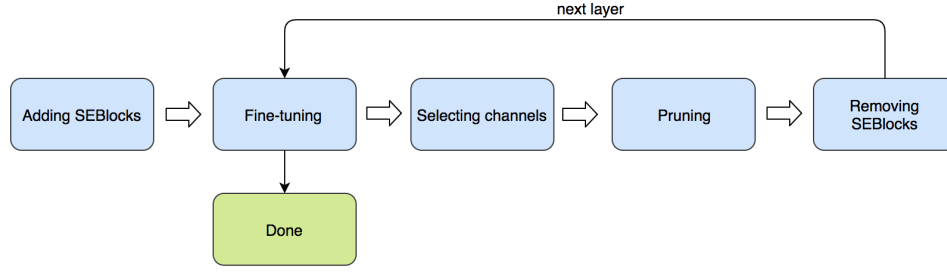


Fig. 3. The overall pruning steps of our method.

to generate scaling factors for channels and adjust scaling factors in backpropagation. In this case, which channel is enhanced or suppressed can be reflected by the scaling factors. Then we can rank channels by their scaling factors with the rule that channels with smaller scaling factors are less important. Because the one-to-one match between each filter and each channel, unimportant filters can be selected according to channel rank.

The attention module (namely SEBlock) used in our method comes from Squeeze-and-Excitation Networks (SENet) [11], which is designed for image classification tasks. SEBlock consists of a global average pooling layer, two fully connected layers, sigmoid activation function and ReLU activation function. SEBlock can be added after convolution layer, take the convolutional layer's output channels as input data, produce scaling factor for each channel.

As shown in Fig. 2, SEBlock takes a C channels as input, obtain a vector with C elements after the global average pooling layer. The number of neurons in first fully connected layer is C/r , where r is a hyper-parameters for controlling module parameter number. Between the first and the second fully connected layer is a ReLU activation function. The second fully connected layer has C neurons in order to produce C output value. Finally, the C output values of are projected between 0-1 by sigmoid activation function to become scaling factors. The output channels of SEBlock can be obtained by multiplying each input channel with corresponding scaling factor.

Although SEBlock was originally designed for improving the accuracy of image classification, its scaling factor reflects the network's choice of feature, which is the embodiment of the channel importance. Channel-level weights introduce more scale features to the network and further enhance the expressive ability of the network. By explicitly describing the importance of inter-layer channels, model would eventually show the phenomenon of restraining or enhancing some channels. Therefore, our method uses the scaling factor produced by SEBlock as channel importance score, and prunes filters in the same layer according to channel rank. Since parameters in SEBlock are randomly initialized, the model would seriously lose its prediction accuracy at first. However, after 1 epoch fine-tuning, the model would quickly return to its original prediction accuracy level, after several epochs it would completely recover from adding SEBlock.

C. Average scaling factor

For SEBlock, scaling factors of channels are data-driven, so we collect a dataset for calculating average scaling factors. Given a collected subset with N images. We calculate average of scaling factor I_C as follows:

$$I_C = \frac{1}{N} F_{se}(X_{<n,H,W,C>}) \quad (4)$$

I_C is a vector having C elements, each element is average scaling factor of corresponding channel, $X_{<n,H,W,C>}$ is C input channels produced by n^{th} image in dataset, W, H is the shape of each channel and F_{se} is the operation of SEBlock.

If the training dataset is not too large, we can directly use the entire training dataset. However, in order to reduce the computational complexity on large datasets, a subset of training dataset is sufficient for calculating average scaling factor. The amount of samples depends on the total number of samples in the training dataset. In our experiments, 10%-20% of the training sample is enough for importance evaluation.

D. Steps of our method

Pruning steps are shown in Fig. 3, the specific steps are as follows:

- 1) **Adding SEBlock.** Given an original model, SEBlock needs to be added after each convolutional layer. Then, we would prune model layer by layer with a predefined pruning rate p , which means p and q in Eq. 3 is equal.
- 2) **Fine-Tuning.** After adding SEBlocks or pruning, the model's prediction accuracy would decrease. By fine-tuning (retraining) the model, it would recover from damage. Meanwhile, SEBlock would scale channels according to their contribution to the loss function.
- 3) **Selecting Channels.** We sort channels in current layer by importance score, and select $p \times C$ channels with small scaling factors as unimportant ones. In our experiment, we set the pruning rate p of each layer to the same.
- 4) **Pruning.** Aiming to remove low-score channels selected in step 3, the filters and convolution kernels corresponding to these channels would be removed as section A describes.
- 5) **Removing SEBlocks** After pruning, in order to compare pruning performance with other pruning methods, we

remove all SEBlocks. In fact, the SEBlock has few parameters(it is mentioned in [10] that SEBlock only leads to about 10% increase in parameter number), which means we could keep SEBlock in practical application.

- 6) **Pruning the Next Layer.** Go to step 2 until all layers are pruned.

E. Pruning Strategy Adjustment

The traditional architecture like AlexNet [21] and VGGNet [22] are often used to verify the effectiveness of pruning methods. In these models, pruning one layer would not change the input shape of other layers except the next layer. But in residual networks like ResNet [14], they have shortcut connections in residual module. Shortcut connection connects the first layer and the last layer in residual module, add them up as the output of residual module. The shortcut connection requires that the shapes of input and output be the same. If the last layer of residual module is pruned, we need to remove the corresponding filters in the first layer without considering their importance score. So it is difficult to prune the last layer of a residual module, in our method, we just prune the first few layers in a residual module, considering that most parameters of residual modules are in these layers.

IV. EXPERIMENTS

We conduct experiments on CIFAR-10 and CIFAR-100 dataset. The CIFAR-10 dataset consists of 60000 images in 10 classes, with 6000 images per class and resolution of each image is 32×32 . The dataset is divided into a training set with 50000 images and a test set with 10000 images. The CIFAR-100 dataset consists of 100 classes, each class contains 500 images for training and 100 images for testing.

On CIFAR dataset, we evaluate our method on two convolutional neural network: VGGNet [13](a variant of vgg16 on the cifar dataset) and ResNet56. The hyper-parameters r mentioned in SEBlock is set to 8 in VGGNet and 4 in ResNet56. For training original model, batch size is set to 128, the learning rate used in first 50 epochs was 0.1, then reduced to 0.01. In pruning process, learning rate is set to 0.01. Weight decay was also used to overcome over-fitting with a coefficient of 0.0001. For calculating average scaling factor, we randomly pick 500 images in each category on CIFAR-10 and 50 images in each category on CIFAR-100. Padding, random cropping and horizontal flipping are applied for data augmentation.

A. Distribution of scaling factors

In this section, the distribution of scaling factors at each layer are visualized. Experiments are conducted on VGGNet with attention module. The distribution of scaling factors are shown in Fig. 4.

The VGGNet contains four stages, distribution of scaling factors in the same stage are similar, so only scaling factors of layer 1,3,5,8 are shown. Fig. 4(a) is the result of the layer 1, which belongs to the first stage. Fig. 4(b) is the result of layer 3, which belongs to the second stage. Fig. 4(c) is the

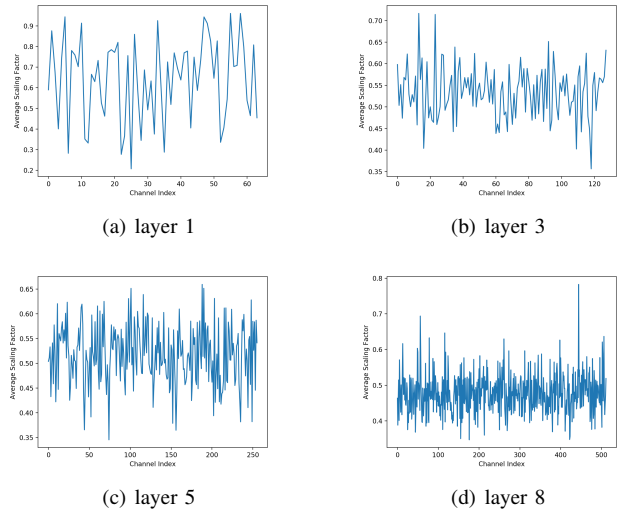


Fig. 4. Distribution of scaling factors.

result of layer 5, which belongs to the third stage. Fig. 4(d) is the result of layer 8, which belongs to the fourth stage. The results show that the distribution of scaling factors in the same layer is quite different, especially the scaling factor between adjacent channels. This phenomenon is similar to the lateral inhibition of human visual neurons, which is beneficial for extracting the shape feature of objects. With the increase of depth, the distribution range of scaling factor decreases, which implies that the importance of channels near the end of neural network are similar.

B. Results on CIFAR-10

On CIFAR-10 dataset, we evaluate our method on VGGNet and ResNet-56, the results are shown in Table I and Table II. The M in each table means million.

TABLE I
VGGNET RESULTS ON CIFAR-10 DATASET.

Model	Filters pruned	Accuracy	Params	FLOPs
Original VGGNet	0%	92.24%	14.98M	6.27×10^8
	30%	92.18%	7.43M	3.07×10^8
Pruned	50%	91.97%	3.82M	1.57×10^8
	70%	91.57%	1.42M	0.57×10^8

As shown in Table I, we prune 30% filters in VGGNet network without obvious accuracy drop, even get an increase in accuracy when we pruned the first few layers. When we prune 50% filters, we still could maintain the model's accuracy with 0.27% accuracy decrease. We conclude that our method correctly measures the importance of the channel which help model recover from pruning. When filter pruning ratio comes to 70%, accuracy decreases more obviously but still within 1% loss. It is worth noting that, by using filter-level pruning method, number of input channels and output channels for each layer would be reduced by 70%, so reduced parameters can be calculated as section III.D describes: $0.7 + (1 - 0.7) \times 0.7 = 91\%$.

TABLE II
RESNET-56 RESULTS ON CIFAR-10 DATASET.

Model	Filters pruned	Accuracy	Params	FLOPs
Original ResNet-56	0%	92.84%	0.85M	2.51×10^8
	30%	92.43%	0.58M	1.76×10^8
Pruned	50%	91.82%	0.42M	1.26×10^8
	70%	90.86%	0.25M	0.76×10^8

Table II shows the results on ResNet-56, different from VGGNet, pruning on ResNet-56 causes a relatively large decrease of accuracy. We prune ResNet-56 with 3 different compression rates as well: 30%, 50%, 70% filters in each layer respectively. Although it cause more serious accuracy decrease than VGGNet, we still prune more than half of the parameters with 1.02% accuracy drop.

According to our analysis, this phenomenon is reasonable because recent network architectures has an improvement of the utilization of parameters. For example, the shortcut connections of ResNet actually makes network structure in a shallow-deep state, which enhance feature fusion and feature delivery. Thus, the parameter utilization of ResNet is much higher than traditional models without shortcut connections, which leads to poor performance on a large percentage of pruning. This phenomenon reminds us that more attentions should be paid when pruning the networks with shortcut connections.

C. Results on CIFAR-100

Our method was compared with Weight Sum [9] and APoZ [10] on CIFAR-100 by pruning VGGNet.

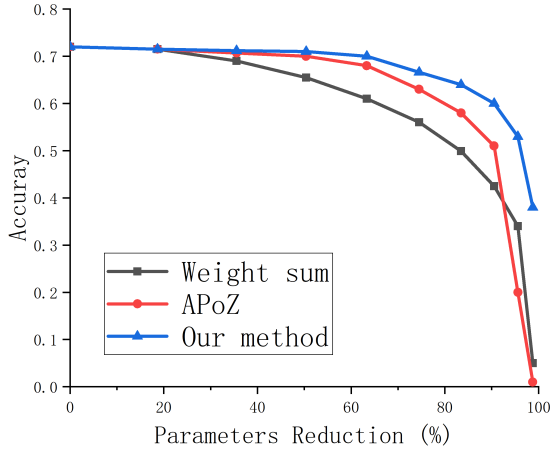


Fig. 5. Our method, Weight sum and APoZ's results on VGGNet.

As shown in Fig. 5, nine groups of experiments were conducted on all methods. Filter pruning ratio of each experiment range from 10% to 90% in step length of 10%. Weight sum has the fastest decline in model accuracy, the accuracy of model begins to decrease significantly when pruning ratio is higher than 10%. It is easy to understand that weight sum

only takes parameter magnitude into consideration, which is not directly related to the model's loss function. APoZ has a better performance, when the pruning rate reaches 50% it begins to have a significant drop in accuracy. This result is reasonable since APoZ considers more further information, channels with more value of zero have less influence on the following layers. However, when pruning ratio is more than 80%, APoZ performs worse than weight sum.

Our method has the best performance among the three methods mentioned above. When pruning rate is less than 50%, the model has almost no drop of accuracy, even if the pruning rate is higher than 50%, the accuracy of the model still decreases slower than the other two methods. An interesting result is that when we prune 90% filters in each layer, our method still retain a certain degree of classification ability, unlike the other two methods whose ability to classify is completely lost. The result indicates that our method correlates the importance of channels with model loss function, which describes the importance of the channels naturally.

D. Results on Mobile Device

The goal of neural network pruning is to reduce the computational resource consumption of neural networks, so as to facilitate deployment on device with limited resources. Therefore, We deployed neural networks on mobile phones for verifying the performance improvement of our pruning method in real scenarios. We tested the performance of the original VGGNet and the VGGNet after removing 50% filters on mobile phones, the device information is: 4 GB Memory, Qualcomm Snapdragon 632 CPU, 2.0GHz basic frequency, training dataset is CIFAR-10 and the framework is Tensorflow. The results is shown in Table III.

TABLE III
RESULT ON MEIZU NOTE 6 MOBILE PHONE.

Parameter	Original	50% filters pruned
Inference time	1046ms	301ms
FLOPs	6.27×10^8	1.57×10^8
Model file size	59.9MB	15.3MB
Parameter Number	14.98M	3.82M
Parameter pruned	0%	75%
Accuracy	92.24%	91.97%

Due to the inference time is affected by hardware, we calculate the average inferences time of classifying a image. Obviously, the pruned model has a significant reduction in file size and inference time. It takes more than 1000 milliseconds for the original model to classify a image, which results in a significant pause of application. On the contrary, the pruned model only takes 301 milliseconds to classify a image, which greatly improves user experience. In addition, the endurance capability of device also benefits from the reduction of computational resource consumption.

V. CONCLUSION

In this paper, we have described a pruning method based on attention mechanism. Different from existing pruning method

based on direct impact of parameters, we use SEBlock to automatically learn the importance of channel during training. As low-score channels and corresponding parameters are removed, memory and computing cost of model would be effectively saved. We validated our method on different datasets, results show that it surpass existing methods under the same pruning ratio. In addition, our method does not require any dedicated libraries or hardwares, thus can be combined with other compression methods.

In future work, we would conduct our method on latest network and larger dataset to verify the generalization of our method. We tend to explore pruning method based on global importance score instead of pruning fixed percentage filters in each layer.

ACKNOWLEDGMENTS

This work was partly supported the NSFC-Guangdong Joint Found(U1501254) and China Information Security Special Fund (NDRC).

REFERENCES

- [1] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.
- [2] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.
- [3] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
- [4] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [5] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. 2015.
- [6] Guorui Zhou, Ying Fan, Runpeng Cui, Weijie Bian, Xiaoqiang Zhu, and Kun Gai. Rocket launching: A universal and efficient framework for training well-performing light net. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [7] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.
- [8] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.
- [9] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *The International Conference on Learning Representations*. MIT, 2017.
- [10] H Hu, R Peng, YW Tai, CK Tang, and N Trimming. A data-driven neuron pruning approach towards efficient deep architectures. arxiv preprint. *arXiv preprint arXiv:1607.03250*, 2016.
- [11] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141. IEEE, 2017.
- [12] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
- [13] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.
- [14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.
- [16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *The International Conference on Learning Representations*, pages 1–14. MIT, 2016.
- [17] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254. IEEE, 2016.
- [18] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2554–2564, 2016.
- [19] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.
- [20] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.
- [21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

An Integrated Software Vulnerability Discovery Model based on Artificial Neural Network

Gul Jabeen*, Junaid Akram*, Luo Ping*, Akber Aman Shah†

*State Key Laboratory of Information Security, School of Software Engineering, Tsinghua University China.

Email: [jgl14,znd15]@mails.tsinghua.edu.cn

Email: luop@mail.tsinghua.edu.cn

†School of Economics and Management, University of Chinese Academy of Science, Beijing, China.

Email: akberaman@hotmail.com

Abstract—Quantitative approaches for software security are needed for effective testing, maintenance and risk assessment of software systems. Vulnerabilities that are present in a software system after its release represent a great risk. Vulnerability discovery models (VDMs) have been proposed to model vulnerability discovery and have been fined to vulnerability data against calendar time. Though, these models have various shortcomings include changes and development of VDMs for different dataset due to diverse approaches and assumptions in their analytical formulation. There is a clear need for an intensive investigation on these models to enhance predictive accuracy of existing VDMs and adopt the actual behavior of software vulnerabilities which were not modeled previously. This study proposed an integrated model to predict a number of software vulnerabilities by hybridizing the Multi-Layer Perceptron (MLP) artificial neural network and Vulnerability Discovery Models. The proposed model is also widely applicable across various vulnerability datasets and models due to its input diversity by providing improved fitting and predictive accuracy. Further, the experimental results show that this model not only retained the properties of traditional parametric VDM models as well as MLP's good nonlinear mapping ability and useful generalization.

keyword Vulnerability discovery model, Artificial neural network, Integrated model, Security, Multi-Layer Perceptron neural network

I. INTRODUCTION

With the development of Internet technology, software vulnerabilities have increased rapidly and caused an increasing number of serious security issues. A critical vulnerability provides an attacker with the ability to access full control of a software [1] [2]. Therefore, a quantitative characterization of the vulnerability discovery rates is necessary to assess the risks associated with the product.

VDMs are the specialization of software reliability growth models (SRGM) that focus on security errors. Nonetheless, most previous studies reveal that these models are based on SRGMs, which are not empirically enough to deal with vulnerabilities of software [3] [4]. VDMs intensive evaluating the security profile of software as compared to the vulnerability predications models because they are focusing only on vulnerable components of software [5] [6] [7] [8] [9] [10]. Software vendors and customers are using accurate VDMs to understand security trends and patch management. However, current research studies are focusing to develop

further improved VDMs to maximize their predictive accuracy. First VDM model (thermodynamic model) was proposed by Anderson, which was based on SRGMs. However, this model is considered as worst in terms of fitting empirical datasets.

Similarly, various statistical models are used either attempting to capture the underlying processes or applying the principles used in other fields of science to discover vulnerabilities. These models are classified into two categories: time-based and effort-based. Time-based models measure the total number of vulnerabilities over time while effort-based models count vulnerabilities based on testing efforts. The current study focuses on time-based models, which still need further investigation to enhance fitting and predictive accuracy of VDMs. Numerous time-based VDMs model are proposed in previous studies i.e., linear model [11], Rescorla's exponential (RE) models [12], Alhazmi and Malaiyas logistic (AML) [3] and multiversion models [13], Weibull model [14], Younis folded (YF) model, Kapur's logistic model [15], Anand and Bhatt's hump-shaped model [16], and Anand multi-version VDM [17], and Sharma's changing point model [18] to model the rate at which vulnerabilities discovered.

All of the above models are proposed to obtain better fitting and predicting models for different vulnerability datasets but most of them are against of certain vulnerability datasets. These models try to get a better model under a certain condition, but it cannot give good results with every vulnerability dataset. Nonetheless, these models have numerous shortcomings as discussed below:

- 1) VDMs uses different approaches with respect to the assumptions and parameters. In this regards, VDMs can predict different vulnerability discovery rates by using the same dataset.
- 2) A single software vulnerability discovery model premised on the constant assumptions and can predict different discovery rates using the same data.

As the neural network can be applied for a variety of areas because without an assumption similar to traditional models, the used model is more universal. In this study, we proposed an integrated approach to solve the aforementioned key challenges of traditional software vulnerability discovery models. The purpose of the study is to provide a flexible

TABLE I: Software Vulnerability Discovery models used in evaluation

Models Name	Model function	Description
Rescorla Exponential (RE) model [19]	$V(t) = N * (1 - e^{-at})$	Exponential model is proposed to fit real data. The number of vulnerabilities discovered at time t decays exponentially with the time.
Alhazmi-malaiya Logistic (AML) Model [3]	$V(t) = \frac{B}{B * C * e^{A * B * t} + 1}$	It is based on the capturing the underlying process of vulnerability discovery and the rate of vulnerability depends on two factors.
Weibull model [14]	$V(t) = \gamma \{1 - e^{-(\frac{t}{\beta})^\alpha}\}$	It assumes that the vulnerability discovery rate varies according weibull probability distribution function.
Younis Folded (YF) Model [20]	$V(t) = \frac{\gamma}{2} [erf(\frac{t-\tau}{\sqrt{2\alpha}}) + erf(\frac{t+\tau}{\sqrt{2\alpha}})]$	It shows vulnerability discovery model based on the folded normal distribution.

method with an accurate representation of data frequencies regardless of their different vulnerabilities discovery rates. An integrated model takes the assistance of ANN (MLP), which serve as a non-linear hybrid system of traditional VDMs. The classical software vulnerability discovery models are used as the base models and the MLP technique is used to combine the results of base models, which helps to eliminate the influence of external parameters and other anomalies of VDMs, arise due to the assumptions made by these parametric models. The proposed model can take the advantage of classical VDMS in an application domain, as in the linear combination model, and the generalization ability of a neural network, and can improve the predictive ability of the software vulnerability assessment models.

The rest of this paper is organized as follows. In Section II, the proposed method is defined in detail. Experimentations and Results analysis are performed in Section III. In section IV, we discuss and highlights the threats to validity. In Section V, we have discussed the related work. Finally, we conclude our work in Section VI.

II. PROPOSED MODEL

The proposed method is elaborated in Fig 1. It is divided into two phases: Phase-I and Phase-II. Detailed elaboration of these phases has been presented below:

A. Phase-I

In the proposed integrated model the results of multiple basic vulnerability prediction models serve as input to the MLP neural network. Therefore, the appropriate base models needed to be selected from many of the VDMs. As vulnerabilities identified in software shows three stages of the S-shaped models [3]. The learning phase is started from the release of the system until the onset of sustained growth because of increasing popularity. It is followed by the linear phase when most of the vulnerabilities are to be discovered. The saturation phase is eventually considered as the last stage. Alhazmi and Malaiya defined mathematically the transition point between different phases for the AML model. Therefore S-shaped models are more accurate than non-S-shaped models for vulnerability discovery process. However, S-shaped models fitting and predictive capability also depends on the skewness in target vulnerabilities, therefore they never give good predictive results for every vulnerability dataset. In this regard, we

have used the four most popular VDMs (i.e., models ranging from an exponential to S-shaped models), which are shown in Table 1, with detailed equations.

After selecting the based VDM models, combined results of these models, are used as input to the MLP neural network. We have used the cumulative number of vulnerabilities as a dependent variable, and time which is measured by months, as an independent variable. A set of known cumulative vulnerability data sequences $v_1(t), v_2(t), v_3(t) \dots v_n(t)$ are used as a input to vulnerability discovery models such as:

$$VDM_i = f(v_1(t), v_2(t), \dots, v_n(t), a_1, a_2, \dots, a_r), (i = 1, 2, \dots, n) \quad (1)$$

where a_1, a_2, \dots, a_n are parameters, and t denots specific vulnerability occurence time. We have used it on a monthly basis. The input vulnerability data sequence is denoted as $v_1(t), v_2(t), \dots, v_n(t)$ and the future trend vulnerability discovery rates can be written as: $v_{n+1}(t), v_{n+2}(t), v_{n+3}(t) \dots v_{n+l}(t)$. After applying the input data sequences $v_1(t), v_2(t), \dots, v_n(t)$ in each vulnerability discovery model (VDM_i). Here, i shows the specific number of VDMs. However, n denoted the total number of vulnerabilities used to estimate parameters and $n + j$ shows the future predicted values.

The output of first VDM_1 approximation solution/fitted data and its future trend or predicted values are determined as $VDM_1(fit) = y_1^{(1)}, y_2^{(1)}, y_3^{(1)} \dots y_n^{(1)}$ and $VDM_1(pre) = y_{n+1}^{(1)}, y_{n+2}^{(1)}, y_{n+3}^{(1)} \dots y_{n+l}^{(1)}$ respectively.

Same process is repeated for the i number of VDMs, which generate the specific outputs regarding the input vulnerability data sequences. After applying the input data sequences $v_1(t), v_2(t), \dots, v_n(t)$ in i^{th} vulnerability discovery models, we determine the output of last VDM approximation solution/fitted data and its predicted values as

$$VDM_i(fit) = y_1^{(i)}, y_2^{(i)}, y_3^{(i)} \dots y_n^{(i)}$$

and

$$VDM_i(pre) = y_{n+1}^{(i)}, y_{n+2}^{(i)}, y_{n+3}^{(i)} \dots y_{n+l}^{(i)} \text{ respectively.}$$

In the first phase the input variable v_1, v_2, \dots, v_n are used to get the estimated and predictive results of different VDMs. Assume that the i^{th} number of VDMs have been used to estimate the vulnerabilities and $y^{(i)}$ denotes estimated and predictive outputs of i^{th} models.

B. Phase-II

In the next stage, the results of vulnerability discovery models have been combined by using multi-layer perceptron.

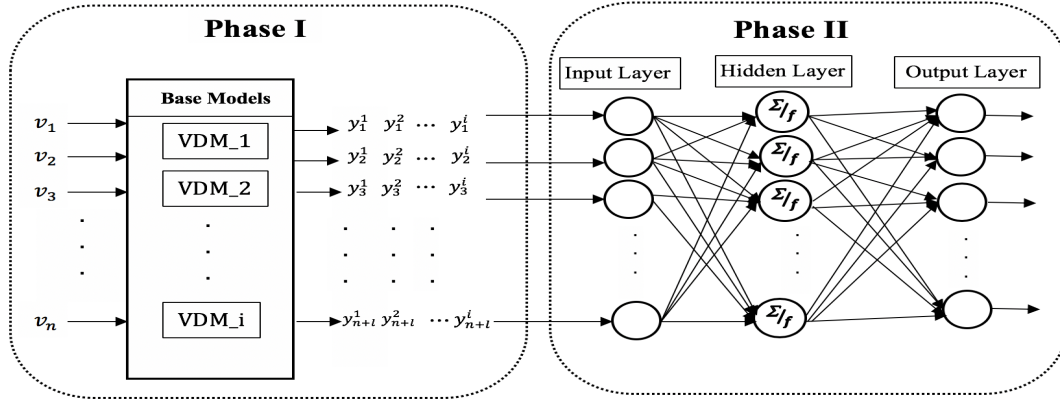


Fig. 1: Detailed diagram of an Integrated VDM model

The MLP is a deep artificial neural network. It is comprised of more than one perceptrons. It works the same way as feed-forward neural network where the back propagation algorithm is in the form of gradient descent function. Almost all algorithms that are applied for MLP training tried to reduce the amount of error by using appropriate functions. In this study, we have used three-layered multilayer perceptron which is composed of an input layer to receive input data and an output layer that predicts input, and in between those two layers, an arbitrary number of hidden layers are present that are the true computational engines of MLP. A nonlinear function is associated with each node like a sigmoid function, except for the input nodes. The MLP network learned a specific target function and adjust weights properly using a general method of linear optimization (gradient function). For this, the derivative of the errors function concerning the network weight is measured. The network weights are changed because of error decreases. The square Euclidean distance is used to compute the error between the actual output and the desired output of a network. The following steps are used, to perform experimentation:

- 1) **Step-1:** The input and output variables are presented first. The vulnerability discovery models outputs from phase 1 are used as input variable ($VDM_1(fit)$, $VDM_2(fit)$, ..., $VDM_i(fit)$) and the actual vulnerability data is used as target variable $v_1(t)$, $v_2(t)$, $v_3(t)$, ..., $v_n(t)$.
- 2) **Step-2:** The entire set of vulnerability data is divided into two parts: training and the testing part. The training dataset is used to train the network for predicting software vulnerabilities. The fitted data of VDMs $y_1^{(i)}$, $y_2^{(i)}$, $y_3^{(i)}$, ..., $y_n^{(i)}$ are used to train the network. For testing the predicted values of different VDMs $y_{n+1}^{(i)}$, $y_{n+2}^{(i)}$, $y_{n+3}^{(i)}$, ..., $y_{n+l}^{(i)}$ are used.
- 3) **Step-3:** The next step in this study is to use 10-fold cross validation method, which divides the data into ten folds. The nine parts are used for training, and the tenth part is used for validation. It is the best validation process to maximize the utilization of vulnerability dataset through

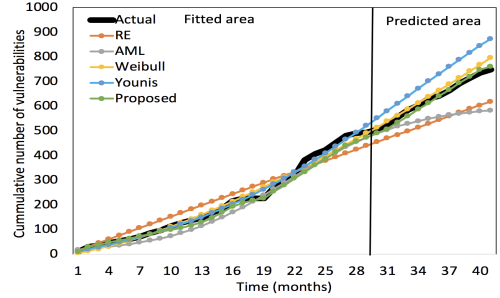


Fig. 2: Models fitting for Window 10

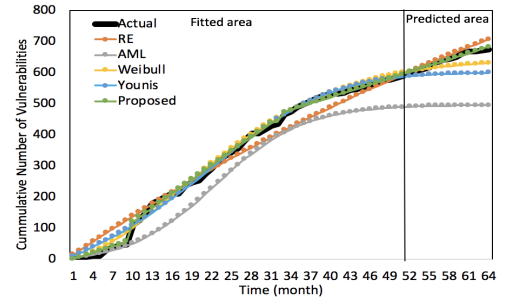


Fig. 3: Models fitting for Internet Explorer

- repeated resampling of the same dataset randomly.
- 4) **Step-4:** After dividing training data into 10-folds, the MLP model is applied for training the models.
- 5) **Step-5:** In this step, the tested data $y_{n+1}^{(i)}$, $y_{n+2}^{(i)}$, $y_{n+3}^{(i)}$, ..., $y_{n+l}^{(i)}$ of VDMs is applied to get the target predicted values $v_1(t)$, $v_2(t)$, $v_3(t)$, ..., $v_n(t)$. The target values are considered as more closer.
- 6) **Step-6:** The statistical efficacy measures are estimated based on the predicted results for all the selected datasets.

III. EXPERIMENTATION AND ANALYSIS

The proposed model has been validated on the security vulnerability data of two software products namely Microsoft

TABLE II: Comparison of VDM models used in evaluation

Models	Windows 10 ($\chi^2_{critical} = 83.6753$)				Internet Explorer ($\chi^2_{critical} = 56.9424$)			
	χ^2	$Pvalue$	DF	R^2	χ^2	$Pvalue$	DF	R^2
RE Model	409.5481	7.467E-52	64	0.9810	428.7739	4.57E-66	41	0.9823
AML Model	1686.6183	0.000E+00	64	0.9739	421.3551	1.33E-64	41	0.9815
Weibull Model	109.5576	3.422E-04	64	0.9938	91.3514	1.06E-05	41	0.9941
Younis Model	245.9183	3.906E-23	64	0.9880	138.7848	1.50E-12	41	0.9913
Proposed Model	34.9911	1.0000	64	0.9990	52.8216	0.102092	41	0.9959

Windows 10 and Internet Explorer obtained CVE details¹. CVE details is a publically available CVE security vulnerability database/information source. Furthermore, a set of models have been used in order to evaluate which model is performing best by comparing their overall performance with the proposed model. We have estimated the parameters of the VDMs using R [21] tool.

A. Model fitting and goodness of fit analysis

To measure the goodness-of-fit, researchers always use Pearson's Chi-square (χ^2) and calculate the statistical value of the curve using the following function.

$$\chi^2 = \sum_{t=1}^n \frac{(O_t - E_t)^2}{E_t} \quad (2)$$

Where O_t and E_t are the observed and expected samples at time t (t^{th} value of the observed sample); E_t denotes the expected cumulative number of vulnerabilities. If the value of χ^2 of VDM for a specific dataset obtains a value less than the corresponding Chi-critical ($\chi^2_{critical}$) value, with the given significant alpha level (0.05) and degree of freedom, it is considered that the model is acceptable. The P-value shows the probability that a statistical value as high as the values obtained by Equation (2) could have occurred by chance. The experimentation data is obtained from the analysis of four VDMs estimated and predicted results. The fitting results of the four VDMs (RE, AML, Weibull and Younis) for both of the software datasets, with there fitted entries based on the χ^2 test P-values.

For Windows 10, the proposed models fit has remarkable improvement than other models in Table II, and the fit is considered very good since the P-value of the χ^2 test is higher than 0.05. Also, the R^2 values are more close to 1 than other single VDMs. From Fig 2, of the fitted vulnerabilities and the predicted vulnerabilities for the selected models, shows clearly that the hybrid approach performing best results than others. Thus, it is required to combine the output results of different models than only depend on single model results. It not only retains the properties of traditional software discovery models but also combines neural network (MLP) good nonlinear mapping ability and useful generalization.

For Internet Explorer, the proposed model also shows good fit than other models, as shown in Table II. The P-value

obtained is 0.102092, which is higher than χ^2 test P-value 0.05. Also, the R^2 values are more close to 1 than others. Fig 3, shows that the fitted and predicted curve of the proposed model has better predictive ability among all other models under consideration.

Thus, it is also concluded that the proposed integrated model can produce good results with different types of vulnerability datasets than the single traditional VDMs. It is also clearly visible in Fig 2 and 3 that the integrated model developed by combining four models have better fitting and end-point predictive capability than the models alone.

TABLE III: Average Bias and Average Errors comparison

	Windows 10		Internet Explorer	
	AB	AE	AB	AE
RE	0.2881	0.9990	-1.7984	1.7984
AML	-2.7989	0.0000	-1.5529	1.5529
Weibull	-0.4353	0.0000	0.3348	0.3348
Younis	-0.9040	0.0000	1.5041	1.5041
Proposed	0.0143	0.0000	-0.1472	0.2443

B. Improved predictive capabilities of VDMs

Goodness-of-fit tests often used to assess the applicability of VDMs. The main use of these models is to predict future trends based on the previous data, rather than reviewing the past behavior. Therefore, predictive capability should also be considered important than just model fitting. The estimated final values for each time point produced by the four existing and proposed integrated VDM are compared with the actual number of vulnerabilities to calculate the predicted errors. The prediction time span is selected as long term, which is the main concern of software users to decide whether the select the software for inclusion in a product with a longer lifetime [22]. Therefore, the last 12 months vulnerability data is predicted for both software. The two normalized prediction capability measures [23], average error (AE) and Average bias (AB), as shown in Equation 3 and 4 respectively, are evaluated.

$$AE = \frac{1}{n} \sum_{t=1}^n \left| \frac{\Omega_t - \Omega}{\Omega} \right| \quad (3)$$

$$AB = \frac{1}{n} \sum_{t=1}^n \frac{\Omega_t - \Omega}{\Omega} \quad (4)$$

In the above equations, n is a total number of time points (in months), and Ω_t is the estimated number of total vulnera-

¹www.cvedetails.com

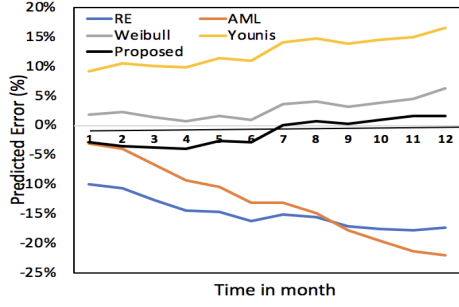


Fig. 4: Predicted Errors of different VDMs for Windows 10

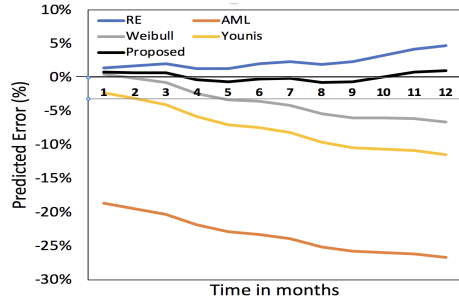


Fig. 5: Predicted Errors of different VDMs for Internet Explorer

bilities at time t , and Ω is the actual number of a total number of vulnerabilities.

The normalized error values ($\frac{\Omega_t - \Omega}{\Omega}$) for windows 10 and Internet Explorer 11 are plotted in Fig 4 and 5. The values of AB and AE are given in Table III. AE always give positive values and AB may give both positive and negative. Fig 4 and 5 shows that the improved models give better predictive results. As in Table 4, the AB and AE values for both yields good results after applying HPEIAM method on each of the models. The results yields after applying our technique give lower AB and AE values than the base models itself.

IV. DISCUSSION AND THREADS TO VALIDITY

In this study, we put forward MLP based integrated vulnerability discovery model, and verify the model's superiority over other models with experimentation. This integrated model is comprised of two phases including the first phase, which reflects the linear combination VDMs, and a second neural network phase, which serve as non-parametric modeling of input data sequences. The proposed approach focuses on the relationship of the performance of VDMs with the specific vulnerability discovery datasets. The fitting capabilities of four VDMs along with the chi-square goodness of fit test as well as R-squared metric indicate that the integrated model fits well, for both of the datasets. Besides, the future trend predictive capability can also be estimated using the two main predictive measuring criterion: AB and AE. The results reveal that the integrated model's predictive results also show more accuracy than the other existing VDMs predictive capability. Therefore, we conclude that the proposed method can be applied as a

solution for software vulnerability rates estimation problems, outperform competing VDMs investigated in this study.

As the proposed integrated model makes the results of multiple basic vulnerability models (base model) as input to the MLP neural network, the accuracy is in part dependent on the predictive accuracy of these models. Therefore the appropriate base models need to be selected from many software vulnerability models. In this study, only four VDMs have been selected as base models, however to analyzing the characteristics of software vulnerability data, and using the appropriate model selection based on specific criterion is required to get more accurate results. Another limitation is the sample data. Our study analyze publicly reported vulnerabilities without considering unreported data. While this is a common drawback among vulnerability research, overcoming this limitation requires direct contact with software vendors.

V. RELATED WORK

Recently, many VDMs have been proposed by researchers, to model the software vulnerabilities accurately. These models either attempting to capture the underlying processes or applying the principles used in other fields of science to discover vulnerabilities. Each model uses a different approach and with different assumptions and parameters.

Among them, the exponential model is designed to fit the real data [12]. In this model, two possible trends were examined, such as the quadratic model and the exponential model. The logarithmic model shows the total number of vulnerabilities as logarithmic growth that was first proposed by Poisson [24] and later it is used by Rahimi [25] adjusting fitting of the model to the vulnerabilities of a specific application. Alhazmi and Malaiya proposed a logistic model called AML model in [26] and analyzed AML in [27]. The predictive capabilities were evaluated in [28] and [4] by using a different set of data. Chan et al. proposed a multi-cycle vulnerability discovery model, which helps to extend the scope of existing models [29]. Younis et al. [20] inspected the applicability of Folded VDM and compared it to AML on Win7, OSX 5.0, Apache 2.0 and IE8, and stated that YF was somewhat better than AML. Joh and Malaiya proposed different S-shaped models based on the distribution of Weibull, normal, beta and gamma distribution to evaluate the applicability of the models using different approaches [30]. Kapur et al. proposed models for the prediction of software vulnerabilities and determine whether software reliability growth models can be used to predict the vulnerability discovery process and show good prediction results [31]. Recently, Anand and Bhatt [16] proposed a hump-shaped model to capture the vulnerability exposure pattern due to the attractiveness of a software product in the market using weighted criteria based ranking approach. Joh and Malaiya analyze vulnerability data using the seasonal index and auto-correlation function approaches, which can be used to improve the vulnerability discovery models [32]. Sharma and Singh proposed a new vulnerability discovery model based on the gamma distribution [33].

Each model defined above uses different approach with different assumptions and parameters. As a result, the VDMs can predict different vulnerability discovery rates using the same data and there is no guidance available about which model should be used in a given situation. This paper attempts to address these problems by integrating the classical VDMs and neural networks.

VI. CONCLUSION

In this study, based on the analysis of the neural network modeling and the linear combination model, we proposed a neural-network-based integrated model. The proposed model achieved better fitting and predictive capability and perform similar or superior as compared to classical and state of art VDM models. It not only retains the properties of vulnerability discovery models but also combines the MLP's good nonlinear mapping ability and generalization. To our knowledge, this is the first study which combines the VDMs and neural network to predict the number of vulnerabilities. Since, the proposed integrated method utilizes the results of classical software vulnerability discovery models serve as input to MLP neural network, so the further study is needed to select appropriate base models from number of software VDMs. In addition, a number of further investigations are possible such as replicate the experiment with advance machine learning techniques and more vulnerability datasets.

VII. ACKNOWLEDGMENT

This research was supported by Beijing National Research Center for Information Science and Technology (BNRist), and National Natural Science Foundation of China under Grant Nos. 90818021, 9071803.

REFERENCES

- [1] C. P. Pfleeger and S. Lawrence, *Security in Computing*. Prentice-Hall, 1997.
- [2] X. Yang, G. Jabeen, P. Luo, X.-L. Zhu, and M.-H. Liu, "A unified measurement solution of software trustworthiness based on social-to-software framework," *Journal of Computer Science and Technology*, vol. 33, no. 3, pp. 603–620, 2018.
- [3] O. H. Alhazmi and Y. K. Malaiya, "Quantitative Vulnerability Assessment of Systems Software," *Reliability and Maintainability Symposium, 2005. Proceedings. Annual*, pp. 615–620, 2005.
- [4] —, "Application of vulnerability discovery models to major operating systems," *IEEE Transactions on Reliability*, vol. 57, no. 1, pp. 14–22, 2008.
- [5] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *ACM Conference on computer and communications security*. Citeseer, 2007, pp. 529–540.
- [6] V. H. Nguyen and L. M. S. Tran, "Predicting Vulnerable Software Components with Dependency Graphs," *Proceedings of the 6th International Workshop on Security Measurements and Metrics - MetriSec '10*, p. 1, 2010.
- [7] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014.
- [8] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.
- [9] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for vulnerability prediction," *arXiv preprint arXiv:1708.02368*, 2017.
- [10] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" *Empirical Software Engineering*, vol. 18, no. 1, pp. 25–59, 2013.
- [11] O. H. Alhazmi and Y. K. Malaiya, "Prediction capabilities of vulnerability discovery models," in *RAMS'06. Annual Reliability and Maintainability Symposium, 2006*. IEEE, 2006, pp. 86–91.
- [12] E. Rescorla, "Is finding security holes a good idea?" pp. 14–19, 2005.
- [13] J. Kim, Y. K. Malaiya, and I. Ray, "Vulnerability discovery in multi-version software systems," in *10th IEEE High Assurance Systems Engineering Symposium (HASE'07)*. IEEE, 2007, pp. 141–148.
- [14] H. Joh and Y. K. Malaiya, "Modeling Skewness in Vulnerability Discovery," *Quality and Reliability Engineering International*, no. September 2013, 2014.
- [15] P. Kapur, N. Sachdeva, and S. Khatri, "Vulnerability discovery modeling," in *International conference on quality, reliability, infocom technology and industrial technology management*, 2015, pp. 34–54.
- [16] A. Anand and N. Bhatt, "Vulnerability discovery modeling and weighted criteria based ranking," *Journal of the Indian Society for Probability and Statistics*, vol. 17, no. 1, pp. 1–10, 2016.
- [17] A. Anand, S. Das, D. Aggrawal, and Y. Klochov, "Vulnerability discovery modelling for software with multi-versions," in *Advances in Reliability and System Engineering*. Springer, 2017, pp. 255–265.
- [18] R. Sharma, R. Sibai, and S. Sabharwal, "Change point modelling in the vulnerability discovery process," in *International Conference on Advanced Informatics for Computing Research*. Springer, 2018, pp. 559–568.
- [19] E. Rescorla, "Is finding security holes a good idea?" *IEEE Security & Privacy*, vol. 3, no. 1, pp. 14–19, 2005.
- [20] A. Younis, H. Joh, and Y. Malaiya, "Modeling learningless vulnerability discovery using a folded distribution," in *Proc. of SAM*, vol. 11. Citeseer, 2011, pp. 617–623.
- [21] R. C. Team, "R development core team. r: A language and environment for statistical computing. r foundation for statistical computing, vienna, austria; 2014," *Google Scholar*.
- [22] F. Massacci and V. H. Nguyen, "An empirical methodology to evaluate vulnerability discovery models," *IEEE Transactions on Software Engineering*, vol. 40, no. 12, pp. 1147–1162, 2014.
- [23] Y. K. Malaiya, N. Karunanithi, and P. Verma, "Predictability of software-reliability models," *IEEE Transactions on Reliability*, vol. 41, no. 4, pp. 539–546, 1992.
- [24] J. D. Musa and K. Okumoto, "A logarithmic poisson execution time model for software reliability measurement," in *Proceedings of the 7th international conference on Software engineering*. IEEE Press, 1984, pp. 230–238.
- [25] S. Rahimi, *Security vulnerabilities: Discovery, prediction, effect, and mitigation*. Southern Illinois University at Carbondale, 2013.
- [26] O. H. Alhazmi and Y. K. Malaiya, "Quantitative vulnerability assessment of systems software," in *Reliability and Maintainability Symposium, 2005. Proceedings. Annual*. IEEE, 2005, pp. 615–620.
- [27] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, "Measuring, analyzing and predicting security vulnerabilities in software systems," *Computers & Security*, vol. 26, no. 3, pp. 219–228, 2007.
- [28] O. H. Alhazmi and Y. K. Malaiya, "Measuring and enhancing prediction capabilities of vulnerability discovery models for apache and iis http servers," in *Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on*. IEEE, 2006, pp. 343–352.
- [29] K. Chan, D. Feng, P. Su, C. Nie, and X. Zhang, "Multi-cycle vulnerability discovery model for prediction," *Journal of Software*, vol. 21, no. 9, pp. 2367–2375, 2010.
- [30] H. Joh and Y. K. Malaiya, "Modeling skewness in vulnerability discovery," *Quality and Reliability Engineering International*, vol. 30, no. 8, pp. 1445–1459, 2014.
- [31] P. Kapur, V. S. Yadavali, and A. Shrivastava, "A comparative study of vulnerability discovery modeling and software reliability growth modeling," in *Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), 2015 International Conference on*. IEEE, 2015, pp. 246–251.
- [32] H. C. Joh and Y. K. Malaiya, "Periodicity in software vulnerability discovery, patching and exploitation," *International Journal of Information Security*, vol. 16, no. 6, pp. 673–690, 2017.
- [33] R. Sharma and R. Singh, "Vulnerability discovery in open-and closed-source software: A new paradigm," in *Software Engineering*. Springer, 2019, pp. 533–539.

Towards Machine Learning for Learnability of MDD tools *

1st Saad Bin Abid

Model-based Systems Engineering (MbSE) Technical University of Munich Model-based Systems Engineering (MbSE)
fortiss GmbH.
Munich, Germany
abid@fortiss.org

2nd Vishal Mahajan

Munich, Germany
vishal.mahajan@tum.de

3rd Levi Lúcio

fortiss GmbH.
Munich, Germany
lucio@fortiss.org

Abstract—Learning how to build software systems using new tools can be a daunting task to anyone new to the job. This is especially true of tools that provide a large number of functionalities and views on the system under development, such as IDEs for Model-Driven Development (MDD). Applying Machine Learning (ML) techniques can help in this state of affairs by pointing out to appropriate next actions to rookie or even intermediate developers. AutoFOCUS3 (AF3) is a mature MDD tool we are building in-house and for which we provide regular tutorials to new users. These users come from both the academia (e.g. students/professors) and the industry (e.g. managers/software engineers). Nonetheless, AF3 remains a complex tool and we have found there is a need to speedup the learning curve of the tool for students that attend our tutorials – or alternatively and more importantly for others that simply download the tool and attempt using it without human supervision. In this paper, we describe a machine learning-based recommendation system named MAGNET for aiding beginner and intermediate users of AF3 in learning the tool. We describe how we have gathered data and trained an ML model to suggest new commands, how a recommender system was integrated in the AF3, experiments we have run thus far, and the future directions of our work.

AF3– MAGNET demo video: <https://tinyurl.com/y5skeeks>

Index Terms—Model-Driven Development (MDD), AutoFOCUS3, Machine Learning, Intelligent Recommendation Systems (IRS), Eclipse IDE, Domain-Specific Languages () development interaction data

I. INTRODUCTION

Modern IDEs are extremely rich in functionality. Environments such as the Eclipse IDE [4] or the whole range of tools proposed by JETBRAINS [6] are complex, offer an ever-increasing amount of functionalities and are highly customizable. Clearly, over the past decade, IDEs have become increasingly competent at their jobs and help the user in a smart manner, by offering intelligent auto-completion, contextual quick fixes and intentions that help with development productivity while lowering learning curves.

For the past 15 years, we have been developing at fortiss the Eclipse-based AF3 tool [14], [28], for building embedded systems. AF3 is mature and stable and includes articulated perspectives for the complete lifecycle of the development

of embedded systems: from requirements, to architecture, deployment, code generation, simulation and verification.

The goal of AF3 is to demonstrate the feasibility, applicability and relevance of MDD tools. It is an open source tool with a 6-month release period and has served and serves as a means to demonstrate state-of-the-art MDD technology. It is also used as a boiler plate to develop proof-of-concept projects with industrial partners [15], [16], [22], [23].

We have recently started offering a tutorial on AF3 [2] to the academic community and to partner or interested companies. The tutorial involves creating and deploying control software onto a real vehicle and implies manipulating different types of software development perspectives (mostly graphical but also textual) via the Eclipse IDE. Given the learning the necessary IDE manipulations requires effort and constant attention and explanations from dedicated human tutors, we have decided to develop an intelligent and non-invasive automated tutor that can be invoked at the press of a button.

Because AF3 offers interconnected visual (model-based) perspectives of the logic of a system under development, providing automated help in such settings differs from doing so for code-based IDEs. Similar tools to AF3 are MATLAB SIMULINK [7] or LABVIEW [10]. AF3 also partly falls in the category of Domain-Specific Language (DSL) workbenches such as MPS [6], METAEDIT+ [9] or ATOM3 [21] – although building DSLs in AF3 must be done programmatically and outside the tool itself, by using EMF facilities.

In this paper we report on our first steps in the usage of ML to aid beginner students in the discovery and understanding of the AF3 tool. In particular, we have implemented an Intelligent Recommendation System (IRS) called MAGNET, that suggests next steps in the context of an AF3 tutorial. The next steps are shown to the user as short videos illustrating the completion of a low-level task, e.g. building states, transitions or simulating the system.

This article is organised as follows: in section II we present the main functionality of the tool, as seen from the point of view of an AF3 developer. Section III details how we have trained and deployed a recommender system in AF3. We then go on to discuss some preliminary results in section V and to place our work regarding the state-of-the-art in section IV. We conclude with future directions in section VI.

II. HIGHLIGHTS

The MAGNET tool is at its core an Intelligent Recommendation System. Core to MAGNET are:

- a *multiclass classification* model trained by using a machine learning algorithm using anonymized sessions of previous students of AF3, and
- recommendation videos that illustrate accomplishing certain tasks in AF3.

The Multiclass classification model in the IRS predicts, with a very short delay, the state of advancement of the student in the tutorial. The prediction is based on the previous interactions of the student with AF3, as well as on the previously learned *multiclass classification* model. Based on the predicted state, the user is then presented with relevant hints that suggest ways of continuing the tutorial – in the form recommendation videos that demonstrate common low-level of AF3. The videos themselves independent from the tutorial and were built by the Human-Centered Engineering department at fortiss [5] having in mind further reuse for an enlarged IRS. The recommendation videos illustrate common modelling tasks such as how to create a component or a state automaton, how to add transitions or action code to the transitions of an automaton or how to simulate a given software model. The aim is that the new users quickly familiarize themselves with the common visual manipulations of the tool while going through the tutorial.

The recommendation videos can be accessed by a user of AF3 in two ways:

- Via a “*Help Me*” button on the AF3 toolbar (as illustrated in figure 1). In this case the IRS proposes three recommendation videos that were top-ranked by the multiclass classification algorithm. After the press of the button, the top ranked video starts playing. If desired, the user can switch to the other two videos in the order in which they were ranked by the IRS. The time taken by the IRS to display the video after the press of the “*Help Me*” is under 2 seconds.
- Through a global help button. In this case the user does not necessarily want the IRS to predict next tasks, but only to provide a full list of possible help videos with common functionalities of AF3. This global help can be accessed via the menu item “Show All Hints” in the AF3 3 tool bar, under the “AI assistance” drop down menu.

We have also implemented the means to activate/deactivate data acquisition directly through the AF3 interface, keeping in mind data privacy and that some students may not want to have their interaction data stored.

III. MACHINE LEARNING PIPELINE

The user-assistance functionality we wish to implement in MAGNET is providing tips to the user based on a predicted next step in the tutorial. In order to achieve this, it is required to predict the next step of the user and then to provide an adequate hint for it, from the set of predefined short videos. The machine learning pipeline we have implemented to achieve

this consist of the following steps: 1) Data collection, 2) & 3) Data processing including data cleaning, feature selection and data labelling, 4) ML model selection consisting of model training, validation and testing and 5) model deployment. This pipeline is depicted in figure 2 and each of its steps is discussed in the subsequent paragraphs.

1) **Data Collection:** In order to acquire the user interaction data, we instrumented the AF3 framework such that interactions of a user with the tool’s graphical user interface are recorded. The instrumentation is achieved using SWT-BOT [13] and does not interfere with AF3’s functionality. Specifically, we have added an SWTBOT plugin on top of AF3. The plugin starts along with the AF3 tool as a server that listens to events that occur in widgets of AF3’s graphical user interface (GUI). Additionally, we have implemented a thin client that communicates via sockets with the SWTBOT server and records to a file all such interactions. The raw data was collected from 11 users during one tutorial session. Altogether, the collected files contain approximately 28000 lines where each line denotes a specific action at a time instance.

2) **Data Cleaning:** Each line in the collected data contains UNIX Epoch time in milliseconds followed by a string for each of the user’s actions/ interaction. The string consists of two sub-parts, a) a GUI contextual description of the user interaction, and b) a general description of the user action initiated via mouse/ keyboard inputs. During analysis of the data, several characteristics of the data were discovered. Some of the actions recorded in the file are correlated with the preceding actions: for instance, closing of a window is mostly followed by default relocation to last visited window. The same trend is observed with respect to few other attributes in the data. Therefore, the data was filtered to remove such correlations and reduce dimensionality, while ensuring minimum loss of significant information. The filtered data consists of approximately 5000 lines.

3) **Feature selection and Labelling:** After having achieved a basic understanding of the data, two features were extracted from the data namely, **Action** (mouse action/ key board action) and **Property** (task dependent description of users state) in a time ordered sequence. A single line of data in isolation at a particular time may be of a practical use from the point of view of task prediction. This is because each line defines a user action at lower/ micro level e.g., one line may indicate opening of a GUI window. On the other hand, a task is defined at a higher level of abstraction which contains lower level actions and conveys a practical meaning such as for example: opening of a window, followed by defining parameters therein and finally moving to a different window perspective. It therefore became essential to label the data in relation to a set of specific AF3 tasks – the classes to be used by the ML classification algorithm. This forms the basis for the machine learning model to understand the lower level data, recognize the current task and predict the next task at an adequate level of abstraction. Nine labels relevant to the tutorial were defined. The labels consist of user tasks such as *navigating to component architecture, creation and specification (definition)*

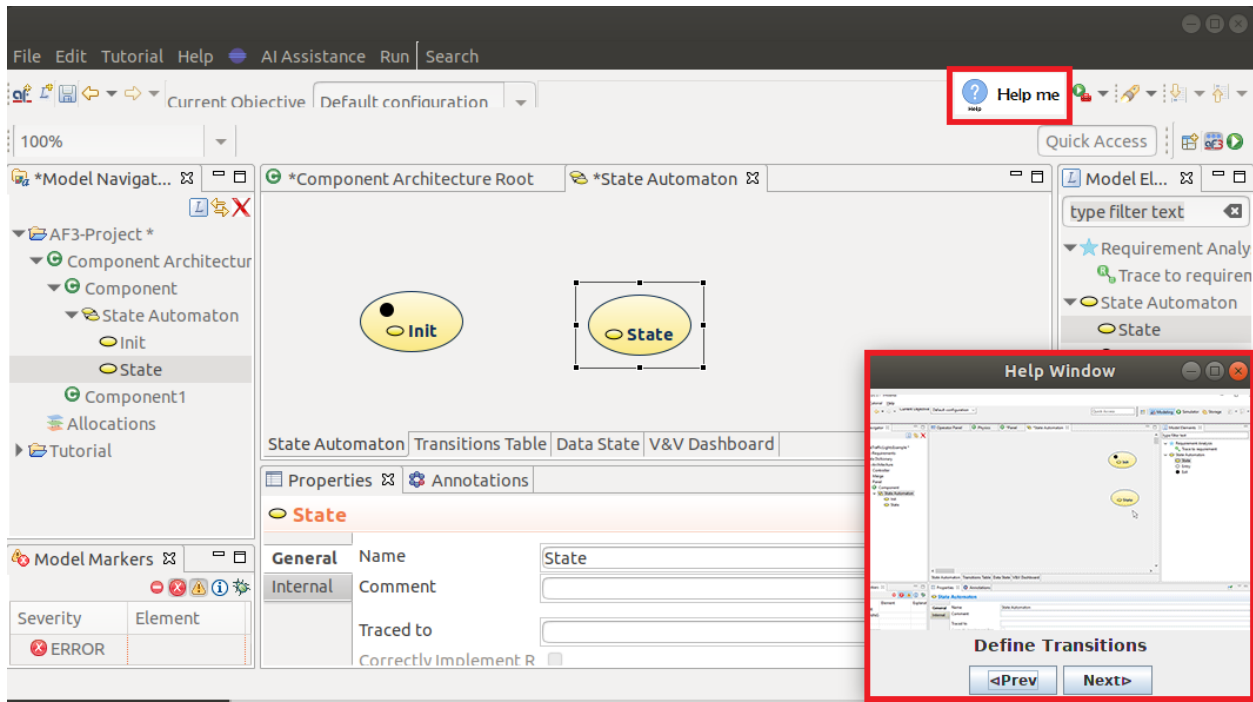


Fig. 1. Recommendation Video in AF3 using “Help Me”

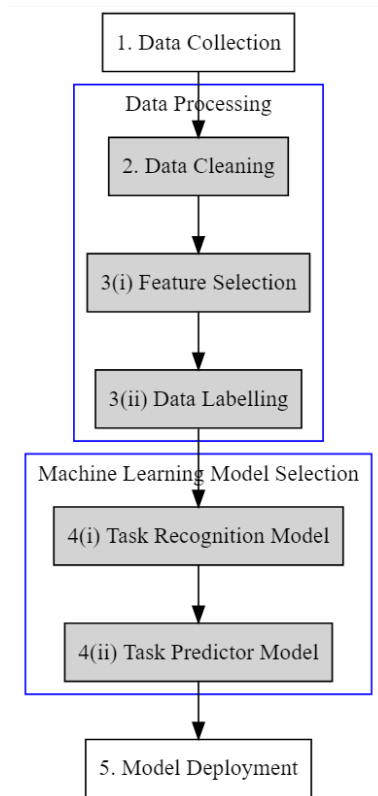


Fig. 2. Machine Learning Pipeline

of components, specification of transitions, creation of ports, writing code and simulation. While defining labels, we kept in mind that these labels should neither be too generic nor too specific, in order to aim for a reasonable granularity that a machine learning algorithm can meaningfully build a model for. The data was labelled manually (approx. 2800 lines) after correlating features in the data with specific tasks. In some cases, when the tasks could not be specifically identified from the features, preceding and succeeding labels were used to infer the current task.

4) **Model Selection:** Learning an objective during the tutorial can be understood as a multi-classification problem based on the features described above. Two levels of abstraction are necessary for predicting the next state of user. The visual representation of the two models is shown in figure 3.

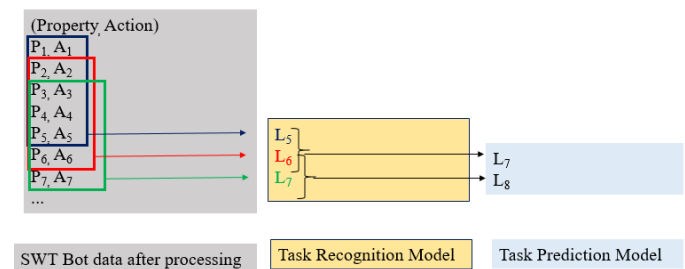


Fig. 3. Visual representation of the models (Recognition and Prediction)

The first level model referred to herein as **Task Recognition Model** takes the input consisting of features: Property (P)

and Action (A) in a time sequence of length u and predicts the corresponding label (L). Figure 3 shows how the **Task Recognition Model** uses a sequence of 5 previous steps i.e., $u = 5$ to predict the current task (label) at a given time t . These predictions act as an input for the second level model. The second model referred herein as **Task Predictor Model** uses the sequence of output labels of length v from Task Recognition Model as an input and predicts the next task of the user at future time step $t + 1$ as an output. As an example, figure 3 shows **Task Predictor Model** uses a label sequence of 2 previous steps i.e., $v = 2$ to predict the label L at future timestep $t + 1$.

Three different machine learning models were tried for training the task recognition model. Due to the similarity of the problem with sequence to sequence prediction, an LSTM model was the first choice [24], [31]. The labelled data was split into training and validation sets in a proportion of 80:20. The LSTM model achieved an accuracy of 20% on the validation data in 100 iterations whereas the training accuracy was very high. The results of LSTM model are shown in figure 4. We attribute the poor performance of the LSTM model on validation data to the model overfitting the training data.

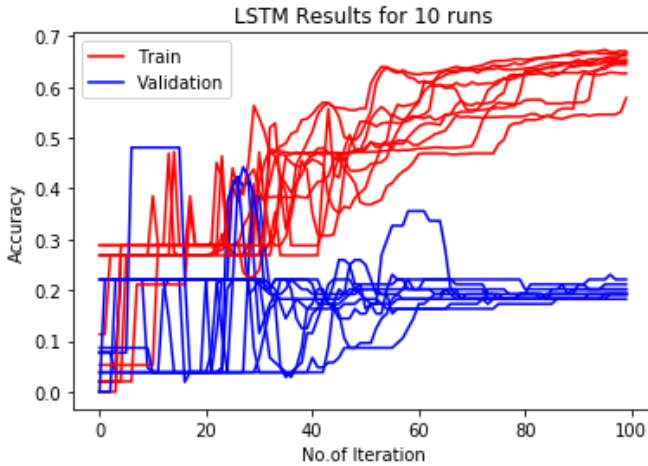


Fig. 4. Training and Validation Accuracy for LSTM Model

Given the weak results obtained from the LSTM, we subsequently moved on to a Random Forest model. This model achieves the best validation and test accuracies of 66% and 58% of all tried models, for $u = 5$. These results show that random forest model outperforms the LSTM model in our setting. To attempt further improvement of the accuracy, we also experimented with XGBoost [18]. The results from XGBoost are comparable with those of Random forest model, with validation accuracy and test accuracy of 65% and 60% respectively. Random forest was selected as the final task recognition model on account of its simplicity in terms of fewer tuning parameters, as compared to XGBoost.

Random Forest is also used for the **Task Predictor Model** and it achieves validation and test accuracies of 51% and 56% respectively for $v = 1$.

5) **Model Deployment:** The complete architecture after deployment is shown in figure 5. Without the MAGNET plugin, the user interacts with AF3 but there is no active guidance or feedback from AF3 (1). If SWTBoT is activated, it records the user interaction continuously in the background (2). The model is deployed as a task predictor plugin in Autofocus. The user clicks on the “Help Me” button and actions the trained machine learning model (3). The data from SWTBoT (4) is processed & analysed by the model to predict the user task(s) (5). The video hints according to the predicted task are displayed on the AF3 interface (6), thus providing active guidance for the user and completing the feedback loop.

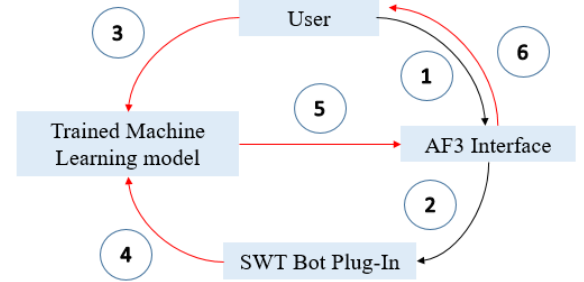


Fig. 5. Architecture of the Recommender System

IV. STATE OF THE ART

The published research that exists focuses on auto-completion mechanism for code-based IDEs, which differ from MDD tools in many ways, chiefly that IDEs for MDD typically deal with many levels of abstraction, often encoded as multiple graphical or textual DSLs. In these tools the process of building software involves mastering a set of different programming paradigms, as well as the inter-relations between different views of the system being developed.

The study of how developers use IDEs was initiated more than a decade ago, at the beginning of the 2000s. The Mylar framework [25] from Kersten and Murphy (later renamed to Mylyn) was one of the first of its kind to provide the developers with the means to define and follow tasks when building software. The framework, distributed as an Eclipse project, allows tracking low-level user commands to keep the state of the task up-to-date while dynamically adapting the IDE such that the navigation possibilities of the IDE are adapted to the developer's needs at each point of the development.

Several authors have concentrated specifically on gathering interaction information from IDEs. The work of Murphy on Eclipse for development in Java [30] was seminal, but more recent efforts exist such as the work from Amann *et al.* for Visual Studio. A survey of methods for collecting IDE interaction data has been proposed by Maalej *et al.* [29].

The natural step after collecting data is to analyse it for patterns that reveal developers' implicit development processes and habitudes, having as goal improving the usability of IDEs. Given the advent of advanced mining tools in the

2010s, authors such as Khodabandelou [26], Damevski [19] and Shepherd [20] have worked on what is now known as *process mining*. The authors have placed importance in not only coming up with models of developer behavior from low-level IDE logs of user-machine interaction, but also in automatically building models of those processes (e.g. using Markov chains) that can be understood by humans.

Since the past 5-10 years researchers have also started exploring how recommender systems can come to the help of software developers. Although the infamous “clippy” that shipped with early versions of the Microsoft office suite was a put off to having recommender systems as part of production software, advances in machine learning and human-computer interaction spawned new attempts at developing such functionality. Autodesk has implemented one such system for AutoCAD to improve the learnability of their tool by contextually proposing previously unseen commands. The company has then conducted a large study with more than 1000 users [27]. The authors of the study conclude that recommender systems are indeed useful and have a rich future in software applications. Damevski [27] as well as Bullmer [17] have explicitly explored classification algorithms to predict developer’s behavior, much as we do. They report accuracies between 20-60%, which are in general lower than the accuracies we achieve. However, it is relevant to mention that the IDEs these authors explore are for code-based development, as opposed to our work on IDEs for model-driven development. In particular, their predictions are made on a large set of commands (e.g. 61 for the study in [17]) for which it will be naturally harder to reach high accuracies as for our work presented here (where 9 labels were used). The same study reports that neural networks have yielded a better accuracy, whereas in the study we present here they have performed the worst.

Outside academia, AF3 strongly relates to modern low-code tools, which employ model- and graphical-based software development principles to enable developers to quickly build, deploy and update applications. The two market leading tools at the time of the writing of this article are *mendix* [8] and the *OutSystems* [11] platform. *mendix* has implemented an AI-based system similar to MAGNET, where most likely steps are suggested to finish a logical workflow. Additionally, a “mentoring” mode is available to teach new developers how to build applications. *OutSystems* is currently investing very strongly in AI-based techniques to aid in software development, having created a laboratory [12] just for this purpose. As with AF3, the AI assistant of the *OutSystems* framework also predicts next steps in the development of the application, which *OutSystems* claims increases developer productivity by 25%. Note that both *mendix* and *OutSystems* are proprietary systems, while AutoFOCUS is distributed as open source.

JetBrains, the developers of a large range of code-based tools for software development, have recently started collecting data to improve their auto-completion systems since 2016 [3]. Additionally, a plugin for predictive coding [1] has been developed for the *IntelliJ* tool from JetBrains that uses machine learning for intelligent auto-completion, as well as

for inserting snippets of code based on comments written in plain English.

V. PRELIMINARY RESULTS

As mentioned in section I, at fortiss we often offer workshops to demonstrate the various functionalities of AF3 to the new users. We have used one of such workshops to evaluate the usefulness of our IRS in one of the AF3 tutorial workshops. The focus group comprised 9 participants and was of mixed technical skills ranging from non-technical (e.g. managers and head of technical departments) to technical (e.g. software engineers). The existing format of the AF3 workshop is such that a human tutor demonstrates the different functionalities of the AF3 at the beginning of the tutorial. The current workshop setting made a few of our recommendation videos redundant for the 3 participants having good programming skills. Nonetheless, the remaining 6 participants having less knowledge of programming (i.e., 5 managers and 1 professor) reported our system was helpful during the workshop giving their lower technical expertise. All feedback from the participants in the study was collected via a formal questionnaire.

As expected, we observed that providing information on AF3 to new users (i.e., technical or non-technical) reduces the use of our IRS. Seen from the reverse perspective, this points towards reducing human intervention in the tutorial (which is desired), which should lead to more usage of the IRS and a better evaluation of our proposal. It is worth mentioning that two participants (1 professor and one software engineer) were very enthusiastic about MAGNET during the workshop, while the others were reticent to using it. This may indicate a polarized view on the usage of MAGNET, although such a claim calls for a future study.

In other interesting (and unexpected) feedback given to us during the workshop, it was suggested having to-the-point videos targeting the logical programming steps related to the tutorial exercise in the workshop. This is reminiscent of our earlier work on building process-aware modelling environments [28], where we proposed a DSL to express static development processes in MDD tools, to help for instance with meeting software certification rules in certain domains (e.g. avionics). Such feedback suggests fusing our previous work on process-aware modelling with the IRS we propose here to allow for both explicit and implicit (machine-learned) software development processes in AF3.

Another point worth discussing is the utilization of *SWT-Bot* [13] for the purpose of gathering the interaction data. The *SWTBot* tool has been built for performing the functional testing of the software interfaces. To the best of our knowledge has not been, up to now, utilized to gather interaction data for machine learning tasks.

VI. FUTURE DIRECTIONS

We have discussed an implementing a recommendation system using ML to ease the learnability of AF3, in particular in a tutorial setting. The results we present are preliminary but encourage us to further pursue this avenue of research. In

particular, are currently considering a number of follow-ups to this work:

- Improving data collection and labelling, as these processes have been up until now long and painstaking. We believe this can be partly achieved by building a “data acquisition” mode in AF3, where users can label sets of their own interactions directly in the tool.
- Focusing specifically on improving user experience by studying how different hint delivery mechanisms (e.g. textual hints or highlighting certain parts of the IDE) improve user satisfaction. The Human-Centered Engineering group at fortiss is currently performing a study on the desirability these mechanisms in AF3. The results of such study should percolate into the MAGNET tool.
- Applying and evaluating different ML techniques to achieve more accurate next-task predictions. The study in this paper indicates that Random Forest achieves good performance – nonetheless these results should be taken with a grain of salt, given the low amount of labels (classes) that data acquisition has been done in the context of a tutorial. When broadening the scope of hints to the whole AF3 tool, it might very well be the case that other machine learning models will perform better.
- Studying how non user interaction data (e.g., AF3-model related data) can improve in task predictions for the new users. For the time being all predictions are based on user/widget interaction and no information about the state of the model of the embedded software system under construction is taken into consideration. Due to its size, it will is not possible to use the complete state of the model for either training or classification. But, by identifying deltas in the state of the model from one moment of time to the next and taking this information into consideration for training and prediction, we believe the accuracy of next task prediction will increase. Note that a system implementing the measurement of such deltas has already been implemented in AF3. We have indeed used this system to evaluate the state of a model in order to locate a developer in a pre-defined development process [28].
- Recalibrating the AF3 tutorial format to maximize the usage of the IRS during upcoming tutorial sessions.
- Extending the number of videos to include help for other parts of the AF3 tool, such as deployment, requirements engineering or formal verification. This is important not only to the IRS, but also as a means to improve the learnability of AF3 in general.

REFERENCES

- [1] AI Predictive Coding plugin. <https://plugins.jetbrains.com/plugin/9203-ai-predictive-coding/>.
- [2] AutoFOCUS3 Tutorials. <https://af3.fortiss.org/docs/tutorials/>.
- [3] Data Collection Post for IntelliJ. <https://blog.jetbrains.com/idea/tag/machine-learning/>.
- [4] Eclipse IDE. <https://www.eclipse.org/ide/>.
- [5] Human-Centered Engineering. <https://www.fortiss.org/en/research/projects/human-centered-engineering/>.
- [6] JetBrains. <https://www.jetbrains.com/>.
- [7] MATLAB Simulink. <https://www.mathworks.com/products/matlab.html>.
- [8] mendix. <https://www.mendix.com/>.
- [9] MetaCase MetaEdit+. <https://www.metacase.com/>.
- [10] National Instruments LabVIEW. <http://www.ni.com/en-us/shop/labview/labview-details.html>.
- [11] OutSystems. <https://www.outsystems.com/>.
- [12] outsystems.ai. <https://www.outsystems.com/ai/>.
- [13] SWTBot. <https://www.eclipse.org/swtbot/>.
- [14] V. Aravantinos, S. Voss, S. Teufl, F. Hölzl, and B. Schätz. Autofocus 3: Tooling concepts for seamless, model-based development of embedded systems. In *ACES-MB&WUCOR@MoDELS*, volume 1508 of *CEUR Workshop Proceedings*, pages 19–26. CEUR-WS.org, 2015.
- [15] S. Barner, A. Diewald, J. Migge, A. Syed, G. Fohler, M. Faugère, and D. Gracia Pérez. DREAMS toolchain: Model-driven engineering of mixed-criticality systems. In *Proc. ACM/IEEE 20th Int. Conf. Model Driven Eng. Lang. Syst. (MODELS '17)*, pages 259–269. IEEE, 2017.
- [16] W. Böhm, M. Junker, A. Vogelsang, S. Teufl, R. Pinger, and K. Rahn. A formal systems engineering approach in practice: An experience report. In *Proc. 1st Int. Workshop Software Engineering Research and Industrial Practices*, pages 34–41, New York, NY, USA, 2014. ACM.
- [17] T. Bulmer, L. Montgomery, and D. Damian. Predicting developers’ ide commands with machine learning. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR ’18, pages 82–85, New York, NY, USA, 2018. ACM.
- [18] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, pages 785–794, New York, NY, USA, 2016. ACM.
- [19] K. Damevski, H. Chen, D. Shepherd, and L. Pollock. Interactive exploration of developer interaction traces using a hidden markov model. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR ’16, pages 126–136, New York, NY, USA, 2016. ACM.
- [20] K. Damevski, D. C. Shepherd, J. Schneider, and L. Pollock. Mining sequences of developer interactions in visual studio for usage smells. volume 43, pages 359–371, April 2017.
- [21] J. de Lara and H. Vangheluwe. ATOM3: A Tool for Multi-formalism and Meta-modelling. In *FASE*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 2002.
- [22] J. Eder, S. Zverlov, S. Voss, M. Khalil, and A. Ipatiov. Bringing DSE to life: exploring the design space of an industrial automotive use case. In *Proc. ACM/IEEE 20th Int. Conf. Model Driven Eng. Lang. Syst. (MODELS '17)*, pages 270–280. IEEE, Sept. 2017.
- [23] M. Feilkas, F. a. P. F. Hölzl, S. Rittmann, B. Schätz, W. Schwitzer, W. Sitou, M. Spichkova, and D. Trachtenherz. A refined top- down methodology for the development of automotive software systems: The keylessentry system case study. Technical Report TUM-I1103, Technische Universität München, 2011.
- [24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.
- [25] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ideas. In M. Mezini and P. L. Tarr, editors, *AOSD*, pages 159–168. ACM, 2005.
- [26] G. Khodabandelou, C. Hug, R. Deneckère, and C. Salinesi. Un-supervised discovery of intentional process models from event logs. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 282–291, New York, NY, USA, 2014. ACM.
- [27] W. Li, J. Matejka, T. Grossman, and G. W. Fitzmaurice. Deploying communitycommands: A software command recommender system case study. *AI Magazine*, 36(3):19–34, 2015.
- [28] L. Lúcio, S. bin Abid, S. Rahman, V. Aravantinos, R. Kuestner, and E. Harwardt. Process-aware model-driven development environments. In *MODELS (Satellite Events)*, volume 2019 of *CEUR Workshop Proceedings*, pages 405–411. CEUR-WS.org, 2017.
- [29] W. Maalej, T. Fritz, and R. Robbes. Collecting and processing interaction data for recommendation systems. In *Recommendation Systems in Software Engineering*, pages 173–197. Springer, 2014.
- [30] G. C. Murphy, M. Kersten, and L. Findlater. How are java software developers using the eclipse ide? *IEEE Softw.*, 23(4):76–83, July 2006.
- [31] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

Assessing the Influence of Size Category of the Project in God Class Detection, an Experimental Approach based on Machine Learning (MLA)

Khalid Alkharabsheh
CiTIUS
Universidad de Santiago de Compostela
Santiago de Compostela, Spain
khalid.alkharabsheh@usc.es

Yania Crespo
Departamento de Informática
Universidad de Valladolid
Valladolid, Spain
yania@infor.uva.es

Manuel Fernández-Delgado
CiTIUS
Universidad de Santiago de Compostela
Santiago de Compostela, Spain
manuel.fernandez.delgado@usc.es

José M. Cotos
CiTIUS
Universidad de Santiago de Compostela
Santiago de Compostela, Spain
manel.cotos@usc.es

José A. Taboada
CiTIUS
Universidad de Santiago de Compostela
Santiago de Compostela, Spain
joseangel.taboada@usc.es

Abstract—Design Smell detection has proven to be an effective strategy to improve software quality and consequently decrease maintainability expenses. In this work, we explore the influence of the size category of the software project on the automatic detection of God Class Design Smell by different machine learning techniques. A set of experiments were conducted with eight different learning classifiers on a dataset formed by 12,588 classes of 24 systems. The results were evaluated using ROC area and Kappa tests. The classifiers change their behaviour when they are used in sets that differ in the value of the selected size information of their classes. This study concludes that it is possible to improve results, mainly in agreement, of God Class detection feeding machine learning classifiers with project size information of the classes to analyze.

Index Terms—Design Smell Detection; Machine Learning; God Class.

I. INTRODUCTION

Software systems available in different departments of organizations to provide many services in different domains. Even though the provided services are essential, but the more critical is the software continues operating without problems and mistakes. Software quality is an important concern for software industries, academic, and researchers. Identifying problems in source code or design of software, correcting and modifying them are some of the main activities of the maintenance process in order to increase quality.

All problems related to the software structure that does not make compile or run-time errors are described by "Design Smell" [11]. The presence of Design Smells negatively affects software quality factors, though technical debt increases [6]. Smells can appear in several software artifacts including variables, instructions, operations, methods, classes, packages, subsystems, layers, and their dependencies.

God Class Design Smell is one of the most detected smells in software according to our systematic mapping study [1]. It has attracted the attention of the research community. God Class is defined with different names as the Large Class Bad Smell [10], the Blob Antipattern [5], and the God Class Disharmony [15]. The term God Class Design Smell is the name we use for bringing both together. It is considered as a class level Design Smell.

Several approaches, tools, and techniques have been proposed to improve the detection of God Class Design Smell. However, some studies evidence the lack of agreement among different tools and human experts. Moreover, they have a set of limitations represented in understanding the precise definition of God Class Design Smell and the process of mapping the definition into effective detection algorithms. The current trend in Design Smell detection has adopted machine learning for deriving detection rules [12], [21].

This paper aims through an exploratory study to investigate whether the size category of the project is relevant in God Class detection. A set of experiments were conducted with eight different classifiers. The selected classifiers are the most recently reported in the literature and they jointly involve all families of classifiers. Results are evaluated using ROC area [4] and Kappa tests [2].

The main contributions of this study can be summarized by determining the influence of project size information on God Class Design Smell detection using machine learning techniques. Moreover, a large dataset for God Class Design Smell detection.

The rest of this paper is organized as follows: Section II describes the related work. Section III presents the problem statement, research question, and hypothesis. Section IV describes the dataset we built. Section V exposes the methodol-

ogy we follow to identify the effectiveness of project domain information in God Class Design Smell detection. Section VI discusses the results of experiments. Section VII presents the main threats to the validity and Section VIII explains our conclusions.

II. RELATED WORK

In this section, we present studies exploiting machine learning techniques for Design Smell detection.

Jochen Kreimer [14] presents a method to detect Design Flaws by combining object-oriented metrics and machine learning. The proposed approach was validated using five Design Flaws: Big Class also known as (The Blob), Feature Envy, Long Method, Lazy Class and Delegator and two software systems: IYC (91 classes) and WEKA (597 classes). Khomh et al. [13] introduce BDTEX (Bayesian Detection Expert) a Goal Question Metric-Based approach to detect Antipatterns using Bayesian Belief Networks (BBNs) stood on rule-based representation. The approach was validated using three Antipatterns: The Blob, Functional Decomposition and Spaghetti Code and two Java programs: Xerces v2.7.0 (589 classes) and GanttProject v1.10.2 (188 classes). Maiga et al. [16] introduce SVMDetect, an approach to detect Antipatterns based on support vector machine. They compute the object-oriented metrics for each class. The empirical study involves four Antipatterns: The Blob, Functional Decomposition, Spaghetti Code and Swiss Army Knife and three open source software: ArgoUML v0.19.8 (1,230 classes), Azureus v2.3.0.6 (1,449 classes) and Xerces v2.7.0 (513 classes). Fontana et al. [9] present their approach based on machine learning techniques for outline the common problems in the previous Design Smells detection. The dataset has formed by the 59,333 classes of 76 systems and a large set of relevant metrics. A set of six code smells: God Class, Data Class, Feature Envy, God Method, Brain Method, and Long Method were detected in the experiment.

As we can see, the related works principally focused on numerical information (mainly object-oriented metrics and other well-known sets of metrics) on classes to detect a set of smells. In our work, we focus the attention on certain nominal project information (size category) and try to explore its influence in God Class detection in order to take this information into account in future work, in the aim of obtaining better results that can be more useful for developers, improving agreement among tools and experts.

The aim of this paper is analyzing if clarifying the size category can lead to variations in the detection. The detection problem is seen as a classification problem; using automatic classifiers that separate classes in having God Class Design Smell or not. Our dataset is formed by the 12,588 classes of 24 systems with different sizes and domains. In order to define our dataset, we analyze these systems using five different tools common in detecting God Class Borland Together [3], PMD [8], iPlasma [18], DÉCOR [20] and JDeodorant [22].

III. PROBLEM STATEMENT

The problem that we examine in this study that most of the research community has not taken into account the impact of nominal project information on Design Smell detection. The suspicion that we have raised involves this type of information can be useful for developers to obtain better detection results. For this reason, in this work, we address the effectiveness and importance of project size nominal information in the detection of God Class Design Smell. To address the study problem, we introduce the following research question:

RQ: Does the differences between the size category of the whole project influence in the detection of God Class Design Smell?

We want to reject the null hypothesis formulated as:

H_0^{size} : Project size does not influence on God Class detection.

We propose to obtain several classifiers trained with and without the project size information under examination, working with the dataset taken as a whole in order to verify whether this kind of information will affect on the behaviour of classifiers or not. If this first exploration succeeds, the next step should be to design the same experiment to investigate the impact of the project size category in detecting God Class as explained in Section V. The behaviour of classifiers is evaluated by ROC area and Kappa performance measurements as we will see in Sections V and VI.

IV. DATASET

Our dataset is formed by the 12,588 classes of 24 open source systems written in Java obtained from SourceForge source code repository which involved different domains and size categories. We selected the SourceForge repository because it is the most widely known and used in the context of open source software, and it supplies useful metadata for projects. The projects are selected according to particular criteria, such as should be written in Java, long life cycle with several version, has a significant change history information (development, maintenance), and the projects should be of different size categories.

The high number of projects and classes is intended to discard the probability of dependencies between classifiers and a particular subset of data. Table I presents the main characteristics of our dataset where includes [Project name, Category of size, Number of Classes, Total Line of Code in the project (TLOCP)].

The dataset is formed as follows: rows x_1 to x_{16} represent numerical attributes (metrics), x_{17} , represent the project size information and x_{18} , the result of classification if a class is God Class or not (resulting from the selected tools as a logical or among them). Table II shows the full list of selected variable x_1 to x_{18} and their definition.

TABLE I
DATASET CHARACTERISTICS

ProjectName	SizeCat.	#Class	TLOCP
jAudio-1.0.4	L	416	117,615
Freemind-1.0.1	L	782	106,396
JasperReports-4.7.1	L	1,797	350,690
Squirrel-1.2	M-L	1,138	71,626
KeyStoreExplorer-5.1	M-L	384	83,144
DigiExtractor-2.5.2	M	80	15,668
AngryIPScanner-3.0	M	270	19,965
Plugfy-0.6	S	28	2,337
Matte-1.7	M-L	603	52,067
sMeta-1.0.3	M	222	30,843
xena-6.1.0	M-L	1,975	61,526
pmd-4.3.x	M-L	800	82,885
checkstyle-6.2.0	M-L	277	41,104
JDitlib-0.3.8	M	78	32,081
JCLEC-4-base	M	311	37,575
Javagraphplan-1.0	S-M	50	1,049
Mpxj-4.7	L	553	261,971
Apeiro-2.92	S-M	62	8,908
FullSync-0.10.2	M	169	24,323
OmegaT-3.1.8	L	716	121,909
Lucene-3.0.0	M-L	606	81,611
Ganttproject-2.0.10	M-L	621	66,540
JFreechart-1.0.X	L	499	206,559
JHotDraw-5.2	M	151	17,807

TABLE II
VARIABLE DEFINITION

<i>Metrics of class and package level (ratio scale)</i>		
Var	Metric	Definition
x_1	LOC	Total Lines of Code
x_2	NCLOC	Non-Comment Lines of Code
x_3	CLOC	Comment Lines of Code
x_4	EXEC	Executable Statements
x_5	DC	Density of Comments
x_6	NOT	Number of Types
x_7	NOTa	Number of Abstract Types
x_8	NOTc	Number of Concrete Types
x_9	NOTe	Number of Exported Types
x_{10}	RFC	Response for Class
x_{11}	WMC	Weighted Methods per Class
x_{12}	DIT	Depth in Tree
x_{13}	NOC	Number of Children in Tree
x_{14}	DIP	Dependency Inversion Principle
x_{15}	LCOM	Lack of Cohesion of Methods
x_{16}	NOA	Number of Attributes
<i>Project level information (nominal scale)</i>		
Var	Information	Values
x_{17}	Size category	L, M-L, M, M-S, S
<i>Smell detection (binary)</i>		
Var	Design Smell	According to
x_{18}	God Class	PMD, iPlasma, JDeodorant, Décor, Together

The chosen projects are analyzed with RefactorIT v2.7 tool¹ [19] to compute a set of important class and package level metrics. The selected metrics are widely used in state of the art [13], [14], [17] for Design Smell detection, and some of them are particularly related to God Class characteristics such as cohesion (Lack of Cohesion (LCOM)), size (Line of Code (LOC)), Number of Attributes (NOA), Number of Children in Tree (NOC), and complexity (Weighted Methods per Class

(WMA). The second part of Table II is regarding projects level information, and we can see the size consists of different categories. We follow the same approach as [7] to classify the projects based on categories.

As can be seen, the nominal data of size categories refers to the size of the Total Line of Code in the whole project (TLOCP). The project size is divided into six categories based on the TLOCP include (Small (S) $\leq 4,999$; $5,000 \leq$ Small-Medium (S-M) $\leq 14,999$; $15,000 \leq$ Medium (M) $\leq 39,999$; $40,000 \leq$ Medium-large (M-L) $\leq 99,999$; $100,000 \leq$ Large (L) $\leq 499,999$; Very large (VL) $\geq 500,000$). In fact, this is more precisely an ordinal scale. But we are not going to treat it as ordinal just as a nominal category.

The final part of the table focused on the selected God Class detection tools. To obtain the value of variable x_{18} , we used the following five Design Smell detection tool: The five tools are DÉCOR, JDeodorant, iPlasma, PMD, and Together. The tools were selected based on the results of a systematic mapping study [1] we conducted on the state of play in the field of Design Smell detection for the period 2000 to 2017. The selected tools are one of the most cited and used in the literature, and are commonly employed in God Class detection. Moreover, all the tools support projects implemented in the Java language and the input source is source code while the output is text in different formats such as CSV, txt, XML. Also, most of the tools focus only on smell detection, except for JDeodorant which includes a refactoring operation that is performed after the Design Smells have been identified. Nearly all of the tools have a GUI except for PMD, which is the only tool that has been developed to support both a Textual User Interface (TUI) and a GUI.

We use the output of these tools considered as experts for feeding the data mining algorithms according to the following criteria. If one tool or more detect God Class Design Smell in a particular class, we assign a true value in the God Class attribute. Otherwise, this attribute is set false (as a logical or among them). According to this strategy, the presence of the God Class smell is distributed along the different size categories of the data we are dealing with. Dataset is available on the web².

Despite the high number of projects, our dataset includes 1,958 God Classes against 12,588 classes. According to our experience in the area, the nature of God Class Design Smell leads to obtaining low ratios of smelly classes detected in each project. As we can observe from Table III, the number of God Classes in all categories are enough as inputs to the classifiers for God Class detection. Our dataset includes only one God Class smell in the small size category (S). In our experience it is normal in a small size project to detect one or at most two God Classes if the project is not a complete disaster. We can either discard this category or add more projects if we find. But we do not, because in this case, the small category do not represent a subset of all dataset or cause a problem if the H_0^{size} . We decide to include this category in our experiments

¹<http://RefactorIT.sourceforge.net>

²<https://citius.usc.es/investigacion/datasets/project-nominal-information>

because we choose all classes in each project to send feedback to the project developers.

TABLE III

BUILDING SET, TESTING SET, AND GOD CLASS DISTRIBUTION BY THE SIZE CATEGORIES. (NP: NUMBER OF PROJECT, NOC: NUMBER OF CLASS, GC: GOD CLASSES)

Category	#NP	#NOC	#GC	%GC
L (Testing set)	6	4,763	967	20%
M-L (Building set)	8	5,123	623	12%
M-L (Testing set)		1,281	132	10%
M (Testing set)	7	1,281	198	16%
S-M (Testing set)	2	112	34	30%
S (Testing set)	1	28	1	4%

The high number of projects in a specific category of the size does not mean having more God classes such as (L, M) categories, but when the number of classes increases in a particular category, it implies increasing the number of God Classes. Therefore, it is difficult to balance the dataset categories. The high number of categories implies a lot of subsets to have God classes distributed across them. We would need to increase the number of projects (and classes) significantly to obtain God Class in all of them. Instead of this, we modify our intention and only will use the M-L size category to build classifiers as explained in the next section.

V. METHODOLOGY

In state of the art for Design Smell detection with machine learning algorithms, a different set of classifiers have been used. We choose a set of eight supervised machine learning techniques that are available in Weka version 3.7 [23], which is a comprehensive collection of machine learning algorithm tools used for data analysis and predictive modeling. The high number of selected techniques allows discarding one of them if it is obtained a particular behaviour or it seems to depend on specific data. We work based on different approaches with the default parameters [NaiveBayes (NB), J48, RandomForest (RF), JRip, SMO, IBK, CHIRP and RandomCommittee (RC)]. We decide training with default parameters in all cases because we are not tuning for obtaining the best classifier. What we want to show is that whatever the classifier is, it is a better classifier if the project size category is taken into account.

Firstly, as an exploratory work, we have trained several classifiers with and without size information (x_{17}) to verify whether the use of this type of information increases the quality of classifiers or not. Secondly, the methodology we propose consists of obtaining a classifier trained with projects with the same value for size nominal variable (x_{17}).

As we stated in Section III, the null hypothesis was formulated as size project information does not affect on God Class detection. The intention is to reject each null hypothesis (H_0^{size}). We assume in our experimental design that if size information is not important, a classifier built for the category Medium-Large (M-L) of size category info (x_{17}), should behave the same when classifying projects from different size categories. If this does not happen and the classifiers behave

worse, we can reject the null hypothesis H_0^{size} and say that the size category information is important.

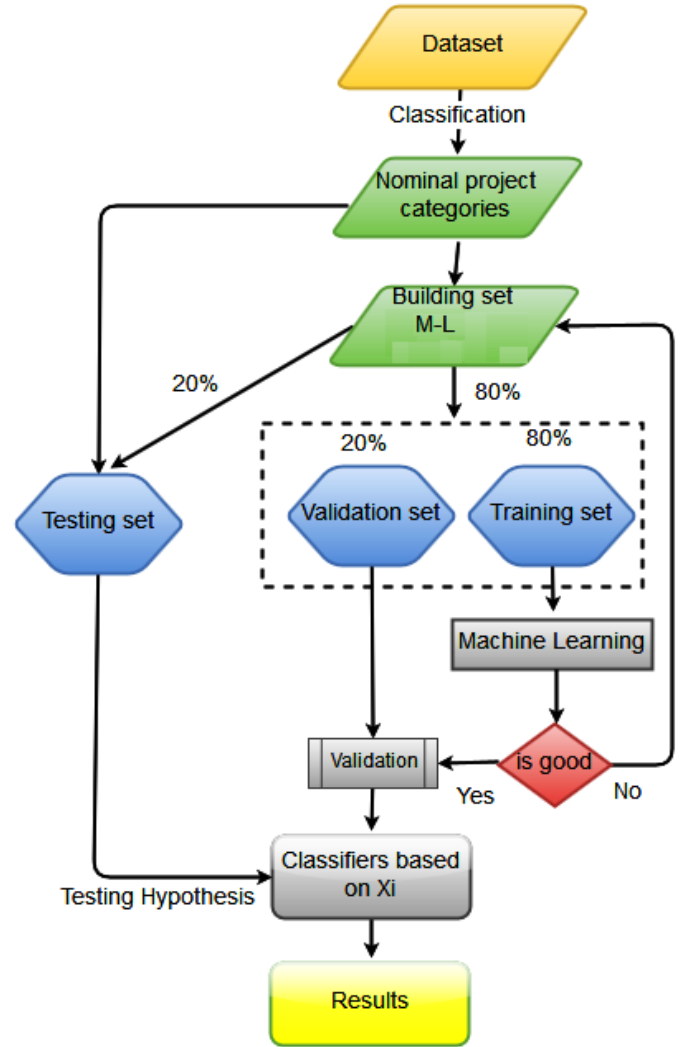


Fig. 1. proposed approach.

Figure 1 summarizes the proposed methodology to obtain the trained classifiers that we will use in the hypothesis testing process. The building set, i.e., the set devoted for building a classifier uses the 80% of the projects in M-L category for x_{17} . The testing set contains the projects in the rest of the categories of size (L, M, S-M, S) plus the remaining 20% from the category that has the highest number of classes (M-L) previously selected to be the building set. This is the set used to test the hypothesis. The set of classes in building and testing data are completely different and are distributed on different project categories.

The building set is divided into two parts. The first part includes 80% to be the training set and the remaining 20% to be the validation set. The training set is supplied to the machine learning algorithm to obtain the required classifier. After that, the validation set is used to validate it. If the classifier is good, the testing set is supplied to the classifier to test the hypothesis. Otherwise, we repeat the process of

using the training and validation set until a good classifier is obtained.

The ROC area test obtains a comprehensive effectiveness evaluation of classifiers. This test shows the relation between the sensitivity and the specificity of the classifiers. Table IV shows the traditional classification of this test.

TABLE IV
INTERPRETATION OF THE ROC AREA (R).

ROC value	Interpretation
$0.5 < ROC \leq 0.6$	Fail
$0.6 < ROC \leq 0.7$	Poor
$0.7 < ROC \leq 0.8$	Fair
$0.8 < ROC \leq 0.9$	Good
$0.9 < ROC \leq 1$	Excellent

Kappa measures the degree of agreement (or concordance) of the nominal or ordinal assessments made by appraisers when assessing the same samples. Kappa can range from -1 to $+1$. The higher the value of Kappa, the stronger the agreement. Table V shows the interpretation of this coefficient.

TABLE V
INTERPRETATION OF THE KAPPA VALUES (K).

Kappa value	Degree of Agreement
$k < 0.20$	Poor
$0.21 \leq k < 0.40$	Fair (Weak)
$0.41 \leq k < 0.60$	Moderate
$0.61 \leq k < 0.80$	Substantial (Good)
$0.81 \leq k \leq 1.00$	Almost perfect (Very Good)

VI. RESULTS AND DISCUSSION

In this section, we present the results of our experiments regarding the influence of the size category nominal information on the detection of God Class to reject the null hypothesis.

Table VI shows the classifiers results when we trained them with all dataset as a whole without size category information (cat. info.) and with size category information. In the ROC area, the majority of classifiers obtain a Good to Excellent behaviour with and without size information except for IBk and SMO in the first case (without size) and JRip in the second case (with size). In particular, SMO's behaviour according to ROC is fair and almost fair in both cases. On the other hand, in the Kappa test, which is considered better indicator because the ratio of detected God Class in the most categories is less than 20%, the classifiers obtain a better result with the selected size information than without this information. In this case, SMO and NB obtained the worst results in general (0.467, 0.4967), respectively. According to Kappa analysis, the behaviour of classifiers is improved with the size information. Based on this, we conducted the experiment explained in Figure 1 on the size in order to analyze H_0^{size} .

Figure 2 shows the ROC area results of the classifiers trained regarding a single category (M-L building set) for project size information x_{17} . All classifiers when exercised with the testing set for the same values of the categories (M-L) for size,

TABLE VI
TRAINED CLASSIFIERS

Classifier	Without size cat. Info.		With size cat. Info.	
	ROC	Kappa	ROC	Kappa
IBK	0.776	0.3062	0.989	0.934
CHAIRP	0.85	0.5505	0.812	0.7041
J48	0.874	0.4572	0.941	0.7936
JRip	0.864	0.529	0.772	0.6097
SMO	0.703	0.5054	0.683	0.467
NB	0.946	0.5659	0.86	0.4967
RC	0.918	0.4799	1	0.9986
RF	0.944	0.5479	1	0.9982

that were used for building the classifiers obtain the highest values of ROC area compared with the rest of the categories in size nominal information (the top black line of the figure). It deserves to say that we are not training the classifiers to be the best. As we stated before, we are just working with default parameters. What we are showing here is that the same algorithm is better being sensitive to the project size category in which it was trained. So, it is considered good evidence that taking into account the information we are analyzing in this work for God Class detection would improve results.

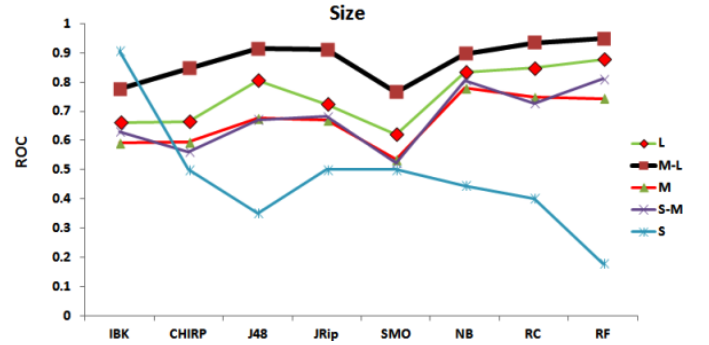


Fig. 2. ROC area performance.

Figure 3 presents the results of the Kappa test for the classifiers trained regarding a single category (building set) for project size information. The testing set with the same value in the size category (M-L) is marked as the black highlighted line in the top of the figure. The better Kappa values are obtained, compared with the rest of the categories except NB in the (M-L) category. Based on the Kappa results, we can say, the size nominal information influence on the detection of God Class. Therefore, we can reject the null hypothesis H_0^{size} .

VII. THREATS TO VALIDITY

In this paper, we highlighted a set of threats to internal validity that affects negatively on our experiment such as the disproportionate number of project in every category of the size nominal information. On the other hand, as threats to external validity, all project is written in Java in order to use the selected set of tools. We did not include the "very large" size projects. Also, the dataset did not include a set of different versions of the same project. This work is focusing on one type of smell (God Class). We need to study other smells that can be detected in different software artifacts. Another threat is

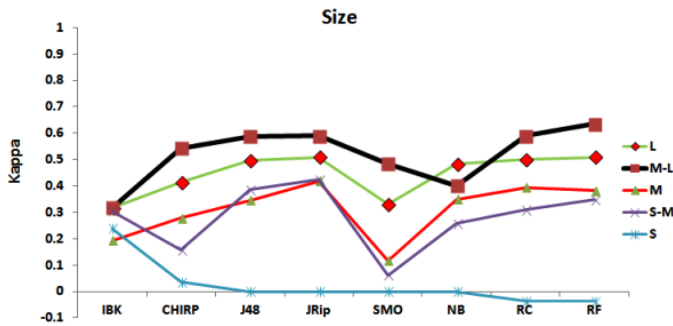


Fig. 3. Kappa values.

the limited number of class level metrics we use and the tools used to analyze the projects (in the metric collection and smell detection).

VIII. CONCLUSIONS AND FUTURE WORK

Our study presented an experimental approach based on machine learning techniques to identify the effective influence of the size category of the project regarding the detection of God Class. In this paper, we present an exploratory study to check whether this information is relevant to be supplied to the classifier and can lead to variations in the classification usefulness, mainly concerning effectiveness (ROC) and agreement (Kappa). All classifiers behave worst according to ROC area and Kappa value tests for categories that were not present when building the classifiers. This study concludes the importance of the size category of the project in the Design Smell detection through machine learning classifiers and should be taken into account in the future works in order to obtain more useful classifiers.

A large dataset formed by the 12,588 classes of 24 systems with different domains and size categories was defined. The classes in the dataset were analyzed by the five different tools selected as experts for God Class detection.

Our future work will focus on replicating the experiments reported in this work rather than using automatic tools as experts we will involve professional human experts from the industry in identifying true God Classes. The same approach could also be extended to study other software context information, such as domain, programming language. Also including other types of design smell, machine learning techniques, and metrics, to confirm whether the proposed methodology has general applicability.

REFERENCES

- [1] Khalid Alkharabsheh, Yania Crespo, Esperanza Manso, and José A. Taboada. Software design smell detection: a systematic mapping study. *Software Quality Journal*, pages 1–80, Oct 2018.
- [2] N.J.M. Blackman and J.J. Koval. Interval estimation for cohen's kappa as a measure of agreement. *Statistics in medicine*, 19(5):723–741, 2000.
- [3] Borland. Together. <http://www.borland.com/us/products/together>. [Accessed: 2014-04-06].
- [4] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.
- [5] William J. Brown, Raphael C. Malveau, Hays W. McCormick III, and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, March 1998.
- [6] Ward Cunningham. The wycash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2):29–30, 1993.
- [7] Francesca Arcelli Fontana, Vincenzo Ferme, Armando Marino, Bohme Walter, and Pawel Martenka. Investigating the impact of code smells on systems quality: An empirical study on systems of different application domains. In *29th IEEE International Conference on Software Maintenance*, pages 260 – 269, September 2013.
- [8] Francesca Arcelli Fontana and Stefano Spinelli. Impact of refactoring on quality code evaluation. In *4th Workshop on Refactoring Tools*, pages 37–40, New York, NY, USA, 2011.
- [9] Francesca Arcelli Fontana, M. Zanoni, Armando Marino, and Mika V. Mantyla. Code smell detection: Towards a machine learning-based approach. In *29th IEEE International Conference on Software Maintenance*, pages 396 – 399, September 2013.
- [10] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Object Technology Series. Addison-Wesley, June 1999.
- [11] Francisco Javier Prez García. *Refactoring Planning for Design Smell Correction in Object-Oriented Software*. PhD thesis, Universidad de Valladolid, Valladolid, 2011.
- [12] A. Hamid, M. Ilyas, M. Hummayun, and A. Nawaz. A Comparative Study on Code Smell Detection Tools. *International Journal of Advanced Science and Technology*, 60:25–32, 2013.
- [13] Foutse Khomh, Stephane Vaucher, Yann-Gaél Guéhéneuc, and Houari Sahraoui. BDTEX: A GQM-based bayesian approach for the detection of antipatterns. *The Journal of Systems and Software*, 84(4):559–572, 2011.
- [14] Jochen Kreimer. Adaptive detection of design flaws. *Electronic Notes in Theoretical Computer Science*, 141(4):117–136, December 2005.
- [15] Michele Lanza and Radu Marinescu. *Object-Oriented Metrics in Practice - Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer, 2006.
- [16] Abdou Maiga, Nasir Ali, Neelesh Bhattacharya, Aminata Saban and Yann Gal Guéhéneuc, Giuliano Antoniol, and Esma Ameur. Support vector machines for anti-pattern detection. In *27th IEEE/ACM International Conference on Automated Software Engineering*, pages 278–281, September 2012.
- [17] Nakarin Maneerat and Pomsiri Muenchaisri. Bad-smell prediction from software design model using machine learning techniques. In *8th International Joint Conference on Computer Science and Software Engineering*, pages 331 – 336, May 2011.
- [18] Cristina Marinescu, Radu Marinescu, Petru Florin Mihancea, Daniel Ratiu, and Richard Wettel. iPlasma: An integrated platform for quality assessment of object-oriented design. In *21st International Conference on Software Maintenance - Industrial and Tool Volume*, pages 77–80, Budapest, Hungary, September 2005.
- [19] Raúl Marticorena, Carlos López, and Yania Crespo. Extending a taxonomy of bad code smells with metrics. In *7th ECCOP International Workshop on Object-Oriented Reengineering (WOOR)*, page 6. Citeseer, 2006.
- [20] Naouel Moha and Yann-Gael Guéhéneuc. DÉCOR: A tool for the detection of design defects. In *22nd IEEE/ACM International Conference on Automated Software Engineering*, pages 527–528, New York, NY, USA, 2007.
- [21] Javier Pérez, Carlos López, Naouel Moha, and Tom Mens. A classification framework and survey for design smell management. Technical Report 2011/01, Grupo GIRO, Departamento de Informática, Universidad de Valladolid, March 2011.
- [22] Nikolaos Tsantalis, Tsantalis Chaikalis, and Alexander Chatzigeorgiou. JDeodorant: Identification and removal of type-checking bad smells. In *12th European Conference on Software Maintenance and Reengineering*, pages 329–331, April 2008.
- [23] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining Practical Machine Learning Tools and Techniques*. Elsevier, 2011.

A Services Development Approach for Smart Home Based on Natural Language Instructions

Yiyan Chen^{1,2}, Zhanghui Liu^{1,2}, Zhiming Huang^{1,2}, Chuanshumin Hu^{1,2}, and Xing Chen^{1,2,*}

¹College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350116, China

²Key Laboratory of Network Computing and Intelligent Information Processing

Fuzhou University, Fuzhou 350116, China

*chenxing@fzu.edu.cn

Abstract—With development of the infrastructures supporting smart home which has entered in a new stage featured by intelligent services. The services based on natural language instructions are aimed at simplifying and improving our lives. To customize and develop these services more efficiently, this paper proposes an approach to model and execute services based on natural language instructions at runtime which introduces the knowledge graph into development process. Firstly, a concept model of knowledge graph is introduced for smart home services. Secondly, we put forward the mechanism to construct the instance of knowledge graph for smart home services. Finally, rules of transformation are provided to achieve mapping from natural language instructions to the services of knowledge graph. We evaluate our model on a prototype system and the experimental results show that our approach can develop services at runtime and effectually reduce the lines of code by 90%.

Index Terms—services development, smart home, knowledge graph, runtime model, natural language

I. INTRODUCTION

The continuous development of smart home infrastructure has ushered in a new era characterized by smart services. A large number of new devices which are complex and heterogeneous such as intelligent robots, smart wearable devices and smart home appliances will access to the network for interaction and collaboration [1] [2]. Human-computer interaction based on natural language has become the main interaction method in smart home like Amazon Echo [3] and Google Home [4].

Smart home service based on natural language commands is actually a process of decision-making and implementation according to the instructions given by users and the information of their environment [5]. At present, the management interface of the device is directly invoked by a programming language such as C or Java to implement services development of smart home. Although this kind of programming has good adaptability, it will bring high price. Developers must be familiar with the management interfaces of different smart devices in order to implement the interaction. In addition, the application system is based on the underlying code bound to a specific intelligent device, which makes it impossible to reuse its management logic. It is hard to expand the manage system

when the devices change, though the management mechanism is similar.

Due to the gap between problem domains and system implementations, accomplishing mapping between them can create significant programming complexity. The knowledge graph is used to describe the concepts, entities, events and relationships between the object, which can serve as a bridge between system requirements and system implementation. Moreover, it turns to be easier to integrate the devices into control system of smart homes. If we want to introduce knowledge graph into the development of smart home services, there are two problems which need to be solved. One is how to define the knowledge graph model to describe the smart home scene. The other is how to map natural language utterances to services provided by devices in the knowledge graph.

In this paper the method is proposed to quickly customize and develop smart home service on the basis of natural language instructions. Our contributions are as following:

- A concept model of knowledge graph is introduced for smart home services.
- We put forward the mechanism to constructing the instance of knowledge graph for smart home services.
- Transformation rules are proposed to achieve mapping from natural language instructions to the services of knowledge graph.

II. RELATED WORK

In the development process of IoT applications, programming is usually carried out at the operating system level. This situation makes the programmer to focus on the underlying related issues rather than the application logic [6]. Guang et al. [7] proposed a top-down IoT application development method and support system. The system could automatically generate platform-specific configuration and execution code by given the platform-independent application management logic. In addition, some research work [8] [9] proposed a service-oriented architecture-based solution that provided a device access interface in the form of RESTful service to shield the heterogeneity and complexity of the underlying system.

The Peking University team conducted research on runtime model theory and construction methods [10] [11]. When the system meta-model and a set of management interfaces

was given, the SM@RT tool [14] can automatically generate code. When the systematic meta-model changed, SM@RT can generate new mapping code automatically.

III. APPROACH OVERVIEW

A. Concept model of knowledge graph

The concept model of knowledge graph provides an abstract representation of smart home services, and aim to describe the concepts and relationships of abstract elements for smart home scene, as shown in Figure 1.

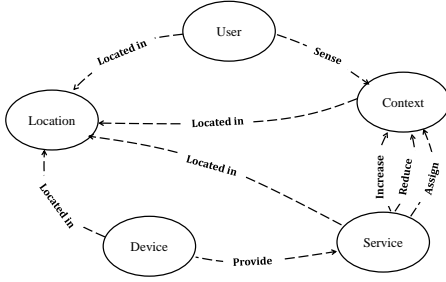


Fig. 1. Concept model of knowledge graph for services of smart home

As Table I illustrates, the concept model defines the concepts of the smart home scenario such as location, user, context, device and service. We give further explanations for those concepts and their properties in this model.

TABLE I
CONCEPTS AND PROPERTIES IN THE CONCEPTUAL MODEL OF SMART HOME KNOWLEDGE GRAPH

Concept Name	Conceptual Properties
Location	<LName>
User	<UName, LName>
Context	<UName, LName, CType, CValue>
Device	<DName, LName, {Key ₁ , Key ₂ , ..., Key _n }>
Service	<DName, LName, DFunction, CType, Effect>

Location indicates a specific area and the name of the area is expressed by *LName*. *User* is used to describe the service object and *UName* is the name of user. The attribute *LName* is the area where the user is located. *Context* represents an environment state of users. *UName* is the name of current service object. *LName* represents the area where the users are located. *CType* is to describe an type of environment status (eg brightness, temperature). *CValue* is the value of status. *Device* represents the equipment object and it contains three properties *LName*, *DName* and *Key_i*. *Key_i* means the configuration parameter or system indicator of device. *Service* indicates a service to change the environment or operating the device and it contains five properties. We need to emphasize the *DFunction* and *Effect* here. *DFunction* is the function interface of the device corresponding to the service. *Effect* is an operation of changing the environment or managing the devices meeting the instruction given by user.

The conceptual model also defines the relationship between concepts mentioned above, as shown in Figure 1. For example,

$X \xrightarrow{\text{Located}} L$ indicates node *User*, *Context*, *Device* and *Service* are located in area *L*. $U \xrightarrow{\text{Sense}} C$ represents user is aware of the state of the context. $S \xrightarrow{\text{Increase}} C$ shows that the service is used to raise the state value of the context.

B. Mapping natural language to service

The implementation of the smart home services in natural language is based on the knowledge graph model of smart home. The goal is to map natural language commands to specific service and the knowledge graph contains all the information of services. Hence, we have two main tasks in this part.

Firstly, the task of information extraction is to identify the semantics in natural language commands. At present, information extraction methods are mainly divided into two categories, one is based on statistical learning method and the other is based on rule pattern matching method. Statistical learning method is portable and robust. However, it needs a lot of training data to set parameters and optimize the system. Rule-based matching method has high efficiency and accuracy. The text of smart home domain has the characteristics of domain, regularity and simple logic. The rule-based method can achieve better extraction effect.

Secondly, the conversion rules are builded to map the information to services of knowledge graph. When services matching the commands are found, it is available to execute the device functional interfaces satisfied with users' need. The specific transformation rules are given in Section V.

IV. RUNTIME MODELING

In this part, we will introduce the runtime modeling method of knowledge graph for smart home. With the purpose of describing the context knowledge of smart home, we establish knowledge graph instances by constructing concept instances and relationship instances.

A. Concept instances modeling

The concept instances are constructed based on the developer configuration. For realizing bidirectional synchronization between concept instances and the real-time information of the scene, a set of rules for mapping are proposed. Developers need to provide relevant configurations to describe specific objective facts in the scenario, including scene information, the mapping of concept instances to smart devices and the environment of service objects.

The configuration of scene information provides a collection of locations $L = \{L_1, L_2, \dots, L_n\}$. The mapping between concept instances and devices provides a collection of devices $D = \{D_1, D_2, \dots, D_n\}$, a collection of services $S = \{S_1, S_2, \dots, S_n\}$ and the relationship between them. The configuration of users' environment provides data which include collection of users $U = \{U_1, U_2, \dots, U_n\}$, positioning devices $T = \{T_1, T_2, \dots, T_n\}$, and environmental state of users $C = \{C_1, C_2, \dots, C_n\}$. Then the corresponding concept instances are generated by the collections mentioned. Moreover,

TABLE II
MAPPING RULES FOR MODEL OPERATIONS OF LOCATION, USER AND CONTEXT INSTANCES

	Location	User	Context
List	$List * L \rightarrow \{L_1, L_2, \dots, L_n\}$ $Get L_i.properties \rightarrow L_i.properties$	$List * U \rightarrow \{U_1, U_2, \dots, U_n\}$ $Get U_i.properties \rightarrow U_i.properties$	$List * C \rightarrow \{C_1, C_2, \dots, C_3\}$ $Get C_i.properties \rightarrow C_i.properties$ $Get C_i.UName \rightarrow C_i.UName$
Get	$Get L_i.LName \rightarrow L_i.LName$	$Get U_i.UName \rightarrow U_i.UName$ $Get U_i.LName \rightarrow \mathbf{RTModel}(Get T_i.location)$	$Get C_i.LName \rightarrow Get U_j.LName \left((\exists U_j) U_j \xrightarrow{Sense} C_i \right)$ $Get C_i.CType \rightarrow C_i.CType$ $Get C_i.CValue \rightarrow Get D_j.key_n \left((\exists D_j) D_j \xrightarrow{Provide} S_i \right)$
Set	-	-	-

TABLE III
MAPPING RULES FOR MODEL OPERATIONS OF DEVICE AND SERVICE INSTANCES

	Device	Service
List	$List * D \rightarrow \{D_1, D_2, \dots, D_n\}$ $Get D_i.properties \rightarrow D_i.properties$	$List * U \rightarrow \{U_1, U_2, \dots, U_n\}$ $Get S_i.properties \rightarrow S_i.properties$
Get	$Get D_i.DName \rightarrow D_i.DName$ $Get D_i.LName \rightarrow D_i.LName$ $Get D_i.key_m \rightarrow \mathbf{RTModel}(Get D_i.key_m)$	$Get S_i.DName \rightarrow S_i.DName \left((\exists D_j) D_j \xrightarrow{Provide} S_i \right)$ $Get S_i.LName \rightarrow S_i.LName$ $Get S_i.CType \rightarrow S_i.CType$ $Get S_i.Effect \rightarrow S_i.Effect$ $Get S_i.DFaction \rightarrow S_i.DFunction$
Set	$Get D_i.key_m \rightarrow \mathbf{RTModel}(Set D_i.key_m)$	-

the SM@RT tool [10] [11] is used to construct the runtime model of the smart devices and positioning devices.

The model operation methods of the concept instances mainly include three types, namely *List*, *Get* and *Set*. *List* is used to get all instances of the same type and its properties. *Get* and *Set* are used to read and write the attribute values of instances respectively. In order to maintain the bidirectional synchronization of the concept instances and the scene realtime information, the mapping rules are defined, as shown in Table II and Table III. There are three main mechanisms for the bidirectional synchronization of knowledge graph and the real-time information of scene by configuration information, runtime models and rules respectively. As Table II shown, the attribute *LName* of location L_i is from the configuration like $Get L_i.LName \rightarrow L_i.LName$. And there are similar situations in other instances of concept. The *LName* of the user U and the *Key_m* of the device need to be obtained through the runtime model, such as $Get U_i.LName \rightarrow \mathbf{RTModel}(Get T_i.location)$. The *LName* and *CValue* of the context as well as *LName* of the service need to be obtained by rules, such as $Get C_i.LName \rightarrow Get U_j.LName \left((\exists U_j) U_j \xrightarrow{Sense} C_i \right)$.

B. Relation instances modeling

As illustrated in Table IV, we further define the rules for constructing relation instance to express the relationships between concepts in a smart home scenario. When two instances of a concept meet certain preconditions, the relationship between them is constructed. For instance in Rule 4, if the area *LName* of the service S_i is equal to the context C_j , and *Effect* of S_i is *Increase*, the relationship instance $S_i \xrightarrow{increase} C_j$ is created. In Rule 2 and 3, the relationship *Scence* and *Provide* involve the user's environmental information and the device's features in the real

scenario, so this kind of information needs to be obtained from the developers' configuration data.

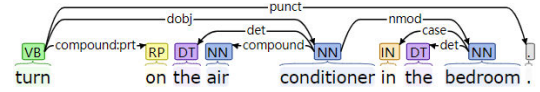


Fig. 2. Instance of dependency parsing tree

V. MAPPING NATURAL LANGUAGE TO SERVICES

In this chapter, we first apply dependency parser to information extraction. Secondly, knowledge reasoning rules are specified to transform the results of information extraction into the search of service nodes in the knowledge graph.

Dependent syntax explains the syntactic structure of sentence by analyzing the dependencies between components within a language unit, as shown in Figure 2. According to the characteristics of commands in smart home scene, we chose dependency parsing to extract the information from natural language instructions.

The final goal is to determine the *Service* instance S_i based on the properties $DName(D)$, $LName(L)$, $CType(C)$ and $Effect(E)$ to invoke the $DFunction$. Therefore, the task of information extraction is to identify four kinds of the mentioned information. This will allow us to convert any natural language utterance to a canonical representation, which we can map to services.

A. Rules of Extraction

In this paper, Stanford Parser is applied for dependency parsing. The generation of the extraction rules mainly depends on the Part of Speech (POS) of the extraction task and the dependency characteristics.

TABLE IV
RULES FOR CONSTRUCTING RELATION INSTANCES

	Relation Instance	Precondition
1	$X_i \xrightarrow{Locate} L_j$	$X_i.LName = L_j.LName$
2	$U_i \xrightarrow{Sense} C_j$	—
3	$D_i \xrightarrow{Provide} S_j$	—
4	$S_i \xrightarrow{increase} C_j$	$S_i.LName = C_j.LName \wedge S_i.CType = C_j.CType \wedge S_i.Effect = "Increase"$
5	$S_i \xrightarrow{Reduce} C_j$	$S_i.LName = C_j.LName \wedge S_i.CType = C_j.CType \wedge S_i.Effect = "Reduce"$
6	$S_i \xrightarrow{Assign} C_j$	$S_i.LName = C_j.LName \wedge S_i.CType = C_j.CType \wedge S_i.Effect = "Assign"$

1) *Extraction Rules of "Effect"*: *Effect* is the operation of changing the environment or managing the device in commands given by users. By summarizing the commands for the smart home scene, we found that an operation is usually represented by a single verb or a verb phrase, like "increase" and "turn on". There is a specific dependency "compound:prt" between words in a verb phrase. The dependency path of the verb part of the sentence is shown in Figure 3. In algorithm 1, we elaborate how to extract the "Effect" according to the dependency relationship. The identifier p is used to represent the nodes in the dependency parsing tree. For example, every word in Figure 2 can be expressed as p . We use $p.word$ to describe content of the node p . $p \rightarrow son$ illustrates the child node of p . The part of speed of p is represented by $p.pos$. $p \rightarrow head$ expresses the parent node of p . Above all, we imply $r(p, q)$ to describe the dependency relationship between node p and q .

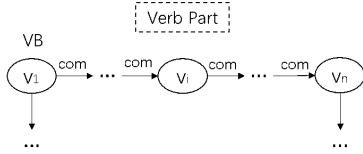


Fig. 3. Dependency paths of verb part

Algorithm 1 Extracting the "Effect" information

Input: Dependency parsing of the command;
Output: "Effect" information E
Find the node p whose part of speech is "VB", $E=p.word$;
while p has a child node $p \rightarrow son$ and the relationship $r(p, p \rightarrow son)$ is "compound:prt" **do**
 $E=E+(p \rightarrow son).word$;
 $p=p \rightarrow son$;
end while

2) *Extraction Rules of DName, LName and CType*: Mapping the attributes *DName*, *LName* and *CType* to the commands is the information of device, area and attribute of environment or device. The expression of these three kinds of information can be summarized in the following two ways.

- Entity of a single noun, like "light", "temperature", "bedroom".

- Entity composed by phrase. It can be divided into two categories. One is the form of "noun+noun" and their relationship is "compound", like "air condition". The other is the form of "adjective+noun", their relationship is "amod", like "sitting room".

The data stored in the knowledge map is matched with the extracted information to classify. In summary, the extraction rules for the three kinds of information as algorithm 2 shows.

Algorithm 2 Extracting the "device", "location" and "attribute" information

Input: Dependency parsing of the command;
Output: "device" information D, "location" information L, "attribute" information A
Find the node p whose part of speech is "NN", $C=p.word$;
while p has a child node $p \rightarrow son$ and the relationship $r(p, p \rightarrow son)$ is "amod" or "compound" **do**
 $N=N+(p \rightarrow son).word$;
 $p=p \rightarrow son$;
end while
Look for knowledge graph node S_i ;
Judge the type of information N is *DName*, *LName* or *CType*.

B. Rules for Knowledge Reasoning

After information extraction is done, we need to map extracted information to service in knowledge graph through transformation mechanism. Universal rules of knowledge reasoning is proposed in term of different extraction results for matching the services. Rules are listed in Table V.

Rule 1, 2 and 3 outline that the user directly specifies the operation of a device. For example, when the user specifies "'D", "L", "C" and "E", construct rule 1. If there is service instance S_i and the value of attribute *LName* is L , *DName* is D , *CType* is C , and *Effect* is E , call the function interface *DFunction* of service S_i .

Rules 4 and 5 describe the knowledge reasoning based on the user's environment when the information given by the user is insufficient to locate the specific service. For example, when the user gives the operation and device, construct rule 4. Firstly, the location information of the device is obtained according to the *LName* of the current user U_i . Then, we

need to find a service S_j that meets the following conditions. The attribute $LName$ value of S_i is equal to $LName$ of U_i , $DName$ of S_i is D , and $Effect$ is E . Finally, the function interface pointed to by the attribute $DFunction$ of the service S_i is called.

VI. EVALUATION

In this section, we build a prototype system of smart home for the actual scene to evaluate the method. Firstly, we evaluate validity of the modeling method for knowledge graph instances. Secondly, we compare the development difficulty and execution performance of the service with the traditional method. Finally, we discuss the precision of information extraction.

A. Study Case

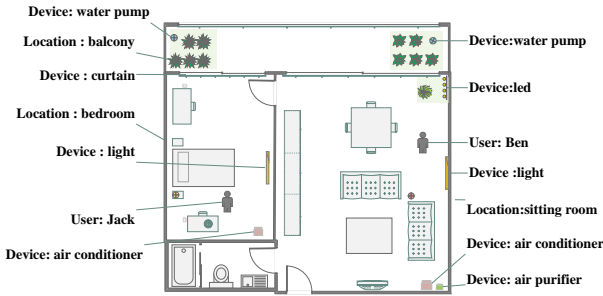


Fig. 4. Example scenario of smart home

There are 3 areas in the scene, which are the sitting room, bedroom and balcony, as shown in Figure 4. For example, there are some smart devices such as an air conditioner, lights and an air purifier in the sitting room. The scene includes four types of environmental states: temperature, humidity, brightness and particulate matter. There are services such as “turn on”, “turn off”, “increase”, “reduce” and so on for various functions of different devices. Fig. 5 shows the instance of the knowledge graph which is generated with respect to the smart home scene.

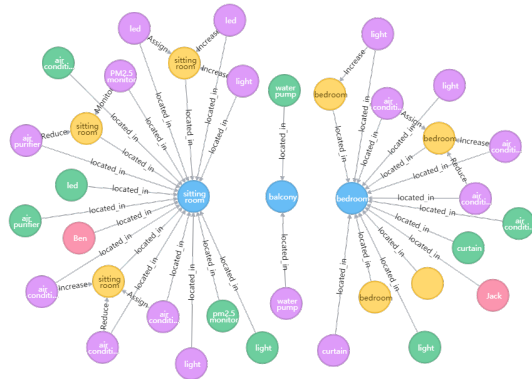


Fig. 5. Instance model of the knowledge graph for smart home

B. Evaluation of the Model

In this section, we develop smart home services based on Java to verify the effectiveness of the method, comparing the process and the difficulty of services development with the method we proposed.

1) *Process of Service Development*: To develop services based on Java, developers must be familiar with the management interfaces of different intelligent devices and realize their interaction. Moreover, because services are based on these management interfaces, their management logic cannot be reused.

We built the runtime model of the smart devices with the help of the SM@RT tool [10] [11], which realizes the data read-write and function invocation of the devices in a unified manner (SM@RT is available from the reference [12]). S-M@RT(supporting model at runtime) is a tool for constructing the runtime software architecture by model-driven, including domain-specific modeling language (SM@RT language) and code generator (SM@RT generator).

SM@RT language allows users to define the meta-model and accessing model of run time software architecture. The meta-model defines the structure and manageable elements of the target system. The accessing model declares the methods of managing these elements in meta-model.

SM@RT generator can automatically generate and maintain the infrastructure of run time software architecture based on meta-model and access model, and reflect the real-time state of the underlying system to the run time model. At the same time, the intelligent device run time model only needs to be constructed once then it can be reused in different smart home scenarios. Therefore, it does not bring extra work.

Based on the run time model of smart devices, developers can automatically build voice control services in smart homes by describing Scenario-Oriented knowledge, configuring the mapping relationship between conceptual instances and smart devices, and describing the environment of service objects.

2) *Difficulty in Service Development*: Table VI compares lines of code for smart home services developed by the two methods. We develop services in Java, and each service is developed independently. When using the method we proposed, developers need to finish the scenario oriented configuration, and the basic code are 115 lines.

For example, it takes 197 lines of code to implement the service S_{11} for temperature adjustment performed by Jack. The workload for these two parts in service S_{11} can be 126 and 71 lines respectively. It only takes 6 lines of configuration code to achieve the same functionality in our method. Implementing these services by traditional method needs 188 lines of code in average, but only 16 lines are required using the proposed method, reducing the workload by 90%.

C. the Accuracy of Information Extraction

We convene 50 volunteers to give natural language instructions based on the prototype system. Because the smart home scene we built is small, the corpus obtained is limited. In the end, we receive 1321 responses regarding what the respondents

TABLE V
RULES FOR KNOWLEDGE REASONING

Condition	Reasoning Rules
D+L+A+O	$(\exists S_i) ((S_i.DName = D) \wedge (S_i.LName = L) \wedge (S_i.CType = A) \wedge (S_i.Effect = O)) \Rightarrow S_i.DFunction$
D+L+O	$(\exists S_i) ((S_i.DName = D) \wedge (S_i.LName = L) \wedge (S_i.Effect = O)) \Rightarrow S_i.DFunction$
L+A+O	$(\exists S_i) ((S_i.LName = L) \wedge (S_i.CType = A) \wedge (S_i.Effect = O)) \Rightarrow S_i.DFunction$
D+O	$(\exists S_j) (\exists U_i) ((U_i.UName = uname) \wedge (S_j.LName = U_i.LName) \wedge (S_j.DName = D) \wedge (S_j.Effect = O)) \Rightarrow S_j.DFunction$
A+O	$(\exists U_i) (U_i.UName = uname) \wedge (\forall S_j) ((S_j.LName = U_i.LName) \wedge (S_j.CType = A) \wedge (S_j.Effect = O)) \Rightarrow S_j.DFunction$

TABLE VI
COMPARISON IN LOC OF TWO APPROACHES

	Basic	S ₁₁	S ₁₂	S ₁₃	S ₂₁	S ₂₂	S ₂₃	S ₃₃	S ₃₄	S ₄₂	S ₄₃	Avarage
Java	0	197	231	181	210	189	140	216	184	140	195	188
Our Approach	115	6	6	6	6	6	6	6	6	6	6	16

want their smart home to do. We extracted the information from these commands and the results of the evaluation are shown in Table VII.

We will discussed the results according to the evaluation data. The accuracy of *DName*, *LName*, *CType* and *Effect* extraction from the table is 93%, 90%, 84% and 80%. Because users have different expressions of the same concept or entity. For example, “sitting room” and “living room”. Therefore, when the user does not use the same word in the knowledge map to perform operations on the device, it will lead to failure of information extraction. The information extraction method of this paper achieves better results when the user follows the scene information.

TABLE VII
THE ACCURACY OF INFORMATION EXTRACTION

Type	Sentence Number	Right Back Number	Precision
DName	867	771	89%
LName	906	815	90%
CType	532	447	84%
Effect	766	612	80%

VII. CONCLUSION

To customize and develop smart home services more efficiently, an approach is proposed to model and execute services based on natural language instructions at runtime. Due to the gap between problem domains and system implementations, accomplishing mapping between them can create significant programming complexity. In order to solve the problem we introduce the knowledge graph serve as a bridge between system requirements and system implementation. Experimental results show that the proposed method can greatly reduce the difficulty and complexity of developing smart home services. But, due to the lack of technical and design capabilities there are some function implementation could be optimized. Firstly, the detection module [13] should be added to check the correctness and completeness of the instance model. Secondly, the information extraction method need to be improved by entity linking to increase the precision.

VIII. ACKNOWLEDGMENT

This work was supported by the National Key R&D Program of China under Grant No. 2017YFB1002000, the Talent Program of Fujian Province for Distinguished Young Scholars in Higher Education and the Guiding Project of Fujian Province under Grant No. 2018H0017.

REFERENCES

- [1] K. Xu, X. L. Wang, W. Wei, H. B. Song, and B. Mao, Toward software defined smart home, *IEEE Communications Magazine*, 2016, 54(5):116-122.
- [2] C. Dixon, R. Mahajan, S. Agarwal, A. J. Brush, B. Lee, S. Saroiu and P. Bahl. An operating system for the home. *Unix Conference on Networked Systems Design and Implementation*, 2012:25-25.
- [3] T. Edwards, J. Edwards, E. Dot. *The Amazon Echo Dot User Guide: Newbie to Expert in 1 Hour! The Echo Dot User Manual That Should Have Come In the Box*. Tim Edwards & Jenna Edwards.
- [4] P. Dempesey. 2017. The teardown: Google Home personal assistant. *Engineering & Technology* 12, 3(Apr. 2017), 80-81.
- [5] G. Campagna, R. Ramesh, S. Xu, F. Michael, S. L. Monica. Almond: The Architecture of an Open, Crowdsourced, Privacy-Preserving, Programmable Virtual Assistant. In *International World Wide Web Conferences Steering Committee*, 2017.
- [6] L. Mottola and G. P. Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *Acm Computing Surveys*, 2009, 43(3):1-51.
- [7] G. Y. Guan, W. Dong, Y. Gao, K. B. Fu and Z. H. Chen. TinyLink: A Holistic System for Rapid Development of IoT Applications. In *International Conference on Mobile Computing and NETWORKING*. New York: ACM, 2017:383-395.
- [8] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. S. D. Souza and V. Trif. SOA-Based Integration of the Internet of Things in Enterprise Services. In *IEEE International Conference on Web Services*. 2009:968-975.
- [9] K. Janowicz, A. Brring, C. Stasch, S. Sachade, T. Everding and A. Llaves. A RESTful Proxy and Data Model for Linked Sensor Data. *International Journal of Digital Earth*, 2013, 6(3):233-254.
- [10] G. Huang, H. Song and H. Mei, SM@RT:towards architecture-based runtime management of Internetwork systems, In *Asia-paci?c Symposium on Internetwork*, pp.1-10, 2009.
- [11] H. Song, G. Huang, F. Chauvel, Y. F. Xiong, Z. J. Hu, Y. C. Sun and H. Mei. Supporting runtime software architecture: A bidirectional transformation based approach. *Journal of Systems & Software*. 2011, 85(5):711-723.
- [12] Peking University. SM@RT: Supporting models at runtime. <http://code.google.com/p/smartv/>, 2009.
- [13] H. He, X. Chen, S. Cai, Y. Zhang, G. Huang. Testing bidirectional model transformation using metamorphic testing. *Information and Software Technology*, 2018, 104:109-129.

Modeling User Contextual Behavior Semantics with Geographical Influence for Point-Of-Interest Recommendation

Dongjin Yu, Kaihui Xu, Dongjing Wang

School of Computer Science and Technology

Hangzhou Dianzi University

Hangzhou, China

yudj@hdu.edu.cn, xkhsy@163.com, dongjing.wang@hdu.edu.cn

Abstract—Point-Of-Interest (POI) recommendation assists users to find their preferred places and helps businesses to attract potential customers. However, the data sparsity and the complexity of user check-in behavior pose a big challenge to POI recommender systems. To tackle this challenge, we propose a POI recommendation method named HeteGeoRankRec based on user contextual behavior semantics. First, to mine the fine-grained user behavioral features, we employ the meta path of Heterogeneous Information Network (HIN) to represent the complex semantic relationship among users and POIs and integrate the context constraints (such as time and weather) into the meta paths. Secondly, we propose a weighted matrix factorization model considering the influence of geographical distance to obtain semantic preference through the user-POI semantic correlativity matrices generated by multiple meta paths. Finally, we introduce a ranking-based fusion method, which unifies the recommendation results of different meta paths as the final preference of users. Experiments on the real data collected from Foursquare show that HeteGeoRankRec has the better performance than the state-of-the-art baselines.

Keywords—location-based social network; heterogeneous information network; context information; point-of-interest recommendation; behavior semantics.

I. INTRODUCTION

In recent years, thanks for the widespread of Internet and mobile devices, Location-Based Social Networks (LBSNs) have become increasingly popular. Users explore their preferred locations, such as libraries, restaurants and stores, through the "check-in" behavior provided by the LBSN services. For example, more than 50 million people use Foursquare every month¹. The personalized POI recommendation service is designed to improve the LBSN service experience by mining user preferences through check-in data.

However, POI recommendation faces serious challenges. First, the number of POIs visited by a user usually accounts for only a small portion of all the POIs, which results in the highly

DOI reference number: 10.18293/SEKE2019-178

sparse user-POI check-in data. In addition, the decision-making process for user check-in behavior is very complex and prone to be affected by rich context information [1]. For example, user's mobility is significantly affected by geographical distance [2,3]. In other words, users are more inclined to visit closer locations. Meanwhile, user's visiting preference might be affected by their social relationships [4], meaning a user may follow the suggestions from his friends or some influential people. Besides, the user's preference may also be affected by the time [5] and the weather [6]. Taking Fig. 1 as an example, Mary may prefer to visit the library on rainy days, while Skye may like to go to the restaurant for lunch. Unfortunately, most existing works lack deep mining of user behavior semantics and suffer from the much worse data sparsity problem.

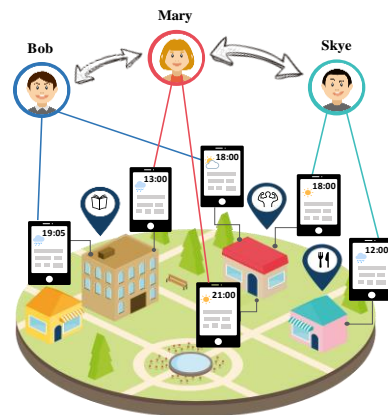


Figure 1. An example of LBSN.

In this paper, we propose a novel POI recommendation model named HeteGeoRankRec, based on user contextual behavior semantics. First, we employ the meta path of Heterogeneous Information Network (HIN) to represent the complex semantic relationships of LBSN. Afterwards, to mine fine-grained user behavioral features, we integrate the context constraints, such as time and weather, to the meta paths. Furthermore, we propose a weighted matrix factorization

¹ <https://foursquare.com/about>

considering geographical distance, from which we obtain the semantic preference through the user-POI semantic correlativity matrixes. Finally, we introduce a ranking-based fusion method, which unifies the recommendation results obtained from different meta paths as the final user preference.

The rest of the paper is organized as follows. After presenting related work in Section II, we introduce the related concepts and problem definition in Section III. Our proposed model is given in Section IV, followed by its experimental evaluation in Section V. Finally, Section VI concludes this paper and outlines the future work.

II. RELATED WORK

The POI recommendation plays an important topic in the field of recommendation systems, attracting the attention from both the academic and industrial fields. The context information, such as geographical influence, has always been regarded as a very significant impact on the recommendation performance [7]. For example, Li et al. [8] considered the user's general interests as a mixture of intrinsic and extrinsic interests, where the former is personal-taste driven and the latter is environment driven. Wang et al. [9] modeled the POI-specific geographical influence between two POIs using three factors: the geo-influence of POI, the geo-susceptibility of POI, and their physical distance. However, only considering the geographical influence is not always enough to represent the user's behavior characteristics.

User's social relationships may affect the user check-in behavior. For example, in [4], Gao et al. held that the social relationships and check-in sequences significantly affect the user's behavior and proposed a fusion model to combine two features to predict user's preference. Besides, Li et al. [10] learned potential locations from three types of friends and incorporated potential locations into matrix factorization model to overcome the cold-start problem. In addition, there are some works considering temporal effect [5] and content information [11]. Although the aforementioned works improve the recommendation performance to some extent by modeling the context information, they lack deep mining of user behavior semantics and suffer from the data sparsity problem.

In recent years, some researches [12,13,14] attempted to apply HIN to the recommendation tasks to integrate more information and represent user behavior semantics. For example, Zhao et al. [13] proposed a HIN-based recommendation method, which uses matrix factorization and factorization machine to solve the information fusion problem. Wang et al. [14] utilized the meta-path-based approach to extract implicit relationships between a user and a POI, and applied logistic regression to establish a prediction model for recommendation. However, they simply regarded the location that the user has not visited as a negative sample, without considering LBSN actually lacks the explicit feedback of POI preferences.

III. THE PRELIMINARY

As an abstract representation of the real world, the information network [15] focuses on the connection between the different types of objects, which is usually defined as follows:

Definition 1. Information Network. An information network is a directed graph $G = (V, E)$, where V is a set of objects and E is a set of links, with an object type mapping function $\Phi: V \rightarrow A$ and a link type mapping function $\varphi: E \rightarrow R$. In other words, each object $v \in V$ belongs to one particular object type $\Phi(v) \in A$, and each link $e \in E$ belongs to one particular relation $\varphi(e) \in R$. When there exists more than one type of object, i.e., $|A| > 1$, or one type of relation, i.e., $|R| > 1$, the network is called a **heterogeneous information network**. Otherwise, it is a **homogeneous information network**.

Definition 2. Network Schema. The network schema is a meta template of information network, denoted as $T_G = (A, R)$, with the object type mapping $\Phi: V \rightarrow A$ and the link mapping $\varphi: E \rightarrow R$.

Fig. 2 shows an example of LBSN heterogeneous information network schema. The network schema serves as a template for a network and tells how many types of objects there are in the network and where the possible links exist, thereby making the heterogeneous information network semi-structured.

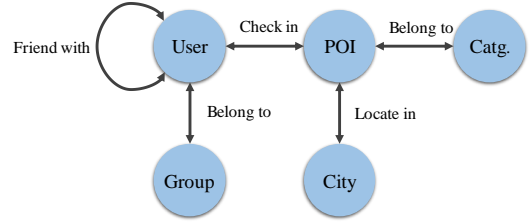


Figure 2. LBSN heterogeneous information network schema.

Definition 3. Meta Path. A meta path M is a path defined on the graph of network schema $T_G = (A, R)$, denoted as $M = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_{l-1}} A_l$.

For simplicity, we denote the meta path as $M = A_1 A_2 \dots A_l$. As shown in Fig. 2, in a LBSN heterogeneous information network, the co-check-in relationship between users can be represented by a meta path $U \xrightarrow{\text{check-in}} P \xrightarrow{\text{checked-in by}} U$, abbreviated as UPU , where U and P represent the user objects and location objects respectively.

Definition 4. Context-constrained Meta Path. A context-constrained meta path is a meta path with the context attribute constraints on relations, denoted as $M_c = A_1 \xrightarrow{\delta_1(R_1)} A_2 \xrightarrow{\delta_2(R_2)} \dots \xrightarrow{\delta_l(R_{l-1})} A_l | S$, where $\delta(R)$ represents a set of context attribute values on relation R , S defines the context of the meta path and the corresponding attribute value constraints.

For example, suppose the whole day is divided into multiple time slices T_1, T_2, \dots, T_n , and the check-in relationship between user U and POI P can occur in multiple time slices. We use $U \xrightarrow{\{T_1, T_2\}} P \xrightarrow{T_1} U$ to indicate that two users check in P at T_1 , and one of them makes a check-in at the T_2 again. Moreover, the path $U \xrightarrow{T_i} P \xrightarrow{T_j} U | \{S: \text{Context} \in \{\text{Time}\}, T_i = T_j\}$ means that two users check in the same POI at the same time slice. Taking Fig. 1 as an example, we can easily find that although three people

all go to the gym, the temporal preferences for Bob and Skye are more similar.

Definition 5. Counting Matrix. For a meta path $M = A_1 A_2 \dots A_l$, we define its counting matrix as $C_M = W_{A_1 A_2} W_{A_2 A_3} \dots W_{A_{l-1} A_l}$, where $W_{A_i A_j}$ is the adjacency matrix between A_i and A_j . The values in the counting matrix represent the number of times the interactions occur between objects.

Problem Definition. Given an LBSN heterogeneous information network G , and a check-in record set S with context information, the problem we try to resolve is to build a personalized recommendation model, and return the Top- K unvisited POIs for each user u .

IV. THE FRAMEWORK

In this section, we present the proposed POI recommendation method, called HeteGeoRankRec, in detail (Fig. 3).

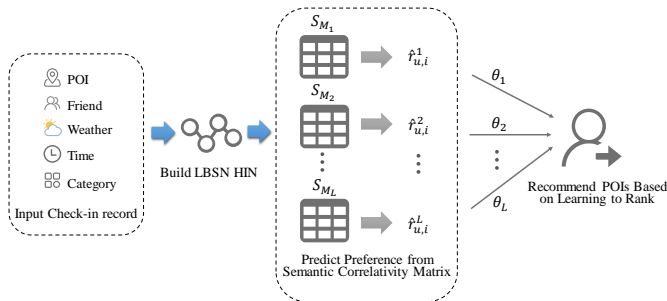


Figure 3. The Framework of HeteGeoRankRec.

A. Build Semantic Correlativity Matrixes Based on Context-constrained Meta Path

1) *Design Meta Paths:* We first employ the meta path to build the semantic relationship sequences for further analysis of user preferences. Taking the path $UPUP$ as an example, it may indicate that users prefer locations where people with common check-in records have checked in, which is a user-based collaborative recommendation. Moreover, the UUP path represents that users prefer the locations checked in by their friends, which is a social recommendation. Therefore, we can make the recommendation more explainable by designing reasonable meta paths to represent different user behavior semantics. Table I lists the meta paths and their corresponding semantics, where C represents the category of POI.

In addition, the context-constrained meta path is used to capture the user's preferences in different contexts (e.g. time, weather). For the meta path like $U * UP$ (e.g. M_3 and M_5 in Table I), we add context constraints as follows:

$$M_C: U \xrightarrow{i} P * P \xrightarrow{j} U \xrightarrow{k} P \{S: Context \in Z, i = j = k\} \quad (1)$$

where $Z = \{Time, Weather\}$ represents a set of context types. Note that weather indicators, such as cloud cover and temperature, can also be divided into multiple numerical

segments. Here, $i = j = k$ indicates that the behavior occurs in same context.

TABLE I. THE META PATHS AND ITS SEMANTICS

Symbol	Meta path	Semantic
M_1	UP	Users prefer locations they have checked in
M_2	UUP	Users prefer locations where their friends have checked in
M_3	$UPUP$	Users prefer locations where people with common check-in records have checked in
M_4	$UPCP$	Users prefer the same category of locations they have checked in
M_5	$UPCPUP$	Users prefer locations where people having checked in the POIs of the same category have checked in

2) *Build Semantic Correlativity Matrix:* We employ the counting matrix defined above as a counting-based correlativity matrix between user objects and location objects, denoted as S_M . This can effectively alleviate the sparsity of the user-POI relation matrix by computing the correlativity through meta path. The counting-based correlativity reflects the idea of high correlativity between nodes with high visibility in the LBSN heterogeneous information network. Such an idea is intuitive and suitable for recommendation task. Taking the time context as an example, the semantic correlativity matrix is built according to Eq. (2), which involves three steps: (a) Divide the time of day into multiple slices T_1, T_2, \dots, T_n , and obtain user check-in records for each slice; (b) Obtain the correlativity matrixes $S_{M_{T_i}}$ by calculating the correlativity from meta paths within each time slice; (c) Add the correlativity matrixes to construct the semantic correlativity matrix S_{M_C} of the context-constrained meta path.

$$S_{M_{T_i}} = (W_{A_1 A_2} W_{A_2 A_3} \dots W_{A_{l-1} A_l})^{T_i}, S_{M_C} = \sum S_{M_{T_i}} \quad (2)$$

B. Predict POI Preference Based on Weighted Matrix Factorization

In this section, we extend the weighted matrix factorization algorithm based on implicit feedback proposed in [16] and optimize the objective function by adding geographical influence factor to make it suitable for POI recommendation.

1) *Calculate User-POI Check-in Probability:* Users are more inclined to visit closer locations. The check-in probability of the user from one location to another x (km) away approximately follows the power law distribution [2], as the following:

$$y = Pr_u(i, j) = a \cdot x^b \quad (3)$$

Let $a = 2^{w_0}$, $b = w_1$, and Eq. (3) is then transformed into Eq. (4) by taking the logarithm:

$$\log y = w_0 + w_1 \log x \quad (4)$$

Let $y' = \log y, x' = \log x$. We then use the linear regression method to optimize the following loss function to obtain the regression coefficient:

$$L = \frac{1}{2} \sum_{n=1}^N (y'_n - p_n)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2 \quad (5)$$

where w_0 and w_1 are regression coefficients, denoted together by \mathbf{w} , p_n is real check-in probability to the x'_n , and the regularization parameter λ is used to prevent the model from overfitting.

Then the check-in probability from POI i to j for user u is normalized by Eq. (6):

$$Pr_u^G(i, j) = \frac{Pr_u(i, j)}{\max(Pr_u)} \quad (6)$$

where the denominator represents the maximum check-in probability of two POIs among the user records.

2) *Incorporate Geographical Influence*: Suppose the corresponding value in the meta-path-based semantic correlativity matrix is represented as $S_{M_{u,i}}$. We define the user implicit preference as follows:

$$r_{u,i} = \begin{cases} 1 & S_{M_{u,i}} > 0 \\ 0 & S_{M_{u,i}} = 0 \end{cases} \quad (7)$$

In other words, $r_{u,i}$ indicates whether there is a value greater than 0 in the correlativity matrix. Furthermore, we introduce $c_{u,i}$ to measure our confidence in $r_{u,i}$. In general, as $S_{M_{u,i}}$ grows, there is a stronger indication that user indeed prefers the location. Eq. (8) defines $c_{u,i}$, where α controls the rate of increase.

$$c_{u,i} = 1 + \alpha S_{M_{u,i}} \quad (8)$$

We believe that the user's preference for unvisited POIs is limited by the distance between the candidate POIs and the POIs that the user has checked in. Thus, based on matrix factorization, the new user preference can be defined as Eq. (9):

$$\hat{r}_{u,i} = \beta x_u^T y_i + \frac{(1-\beta)}{|D_u|} \sum_{k \in D_u} Pr_u^G(i, k) x_u^T y_k \quad (9)$$

where β is geographical influence factor, D_u represents a set of POIs that user u has checked in, x_u and y_i represents the latent feature vectors under same dimension f for user u and POI i .

Then, we solve the low-dimensional feature vector corresponding to the user and the POI by minimizing the loss function defined as Eq. (10) where λ is used to prevent the model from overfitting.

$$\min_{x_u, y_i} \sum_{(u,i) \in T} c_{u,i} (r_{u,i} - \hat{r}_{u,i})^2 + \lambda (\|x_u\|^2 + \|y_i\|^2) \quad (10)$$

The alternating least squares method is used to optimize the above loss function. For simplicity, we define the following variable:

$$\tilde{y}_i = \beta y_i + \frac{1-\beta}{|D_u|} \sum_{k \in D_u} Pr_u^G(i, k) y_k \quad (11)$$

The updating equations for x_u and y_i are obtained as:

$$x_u = (\sum_i c_{u,i} \tilde{y}_i \tilde{y}_i^T + \lambda I)^{-1} \cdot \sum_i c_{u,i} r_{u,i} \tilde{y}_i \quad (12)$$

$$y_i = (\beta^2 \sum_u c_{u,i} x_u x_u^T + \lambda I)^{-1} \times \beta \sum_u c_{u,i} (r_{u,i} x_u + \frac{1-\beta}{|D_u|} \sum_{k \in D_u} Pr_u^G(i, k) x_u^T y_k x_u) \quad (13)$$

C. Recommend POIs Based on Learning to Rank

Suppose we have designed F meta paths and G context-constrained meta paths, and have obtained $L = F + G$ user-POI semantic correlativity matrixes $S_M^1, S_M^2, \dots, S_M^L$. Each matrix generates the user semantic preference $\hat{r}_{u,i}^l$ through the matrix factorization algorithm described above. After combining the different semantic features, the final preference of user u for POI i can be formulated as:

$$r_{u,i}^* = \sum_{l=1}^L \theta_l \cdot \hat{r}_{u,i}^l \quad (14)$$

where θ_l represents the weight of the preference obtained by meta path l .

LBSN often lacks negative feedback, because we regard the POIs that the user has checked in as the positive samples. However, the POIs where the user has not visited yet does not simply mean that they are not interested in (they may not find this location). Therefore, a direct and effective recommendation model should be able to better rank the POI pairs for users, indicating that the user's preference for the POI with high correlativity is greater than the POI with low correlativity in user semantic correlativity matrix. Here, we adopt the idea of pair-wise learning. More specifically, we use the relative orders of POIs as the samples to learn the weights in Eq. (14).

Based on the method proposed in [17], we use the Eq. (15) to express the probability that user u prefers POI i instead of j :

$$p(i >_u j | \theta) = \frac{1}{1 + e^{-(r_{u,i}^* - r_{u,j}^*)}} \quad (15)$$

where $\theta = \{\theta_1, \theta_2, \dots, \theta_L\}$ is a weight vector, $>_u$ represents the ordering relationship of two POIs.

According to the Bayesian formula, if we want all the POIs to be sorted correctly, we need to maximize the following posterior probability:

$$p(\theta | >_u) \propto p(>_u | \theta) p(\theta) \quad (16)$$

Assuming that the user's ranking preference for POI pairs is independent, the likelihood function can be defined by:

$$p(R | \theta) = \prod_{u \in U} p(R_u | \theta) = \prod_{u \in U} \prod_{(i >_u j) \in R_u} p(i >_u j | \theta) \quad (17)$$

where R represents a set of ordering relationships of the POI pairs.

We assume that $p(\theta)$ follows a Gaussian distribution with zero mean and variance-covariance matrix $\sum_{\theta} = \lambda_{\theta} I$. Thus, the objective function of ranking optimization can be formulated as:

$$\begin{aligned} O(\theta) &= -\ln p(\theta | >_u) = -\ln p(>_u | \theta) p(\theta) \\ &= -\sum_{u \in U} \sum_{(i >_u j) \in R_u} \ln p(i >_u j | \theta) - \lambda_{\theta} \|\theta\|^2 \end{aligned} \quad (18)$$

We employ stochastic gradient descent (SGD) to optimize the above objective function. After obtaining θ , the predicted value of user u for all POIs can be calculated by the Eq. (14). Finally, we select K POIs that user has not visited with the highest predicted value and recommend them to the user.

V. EXPERIMENTS

A. Experimental Setup

1) *Dataset*: The experiments are based on the Foursquare dataset² provided by the author of literature [10], including the real-world check-in data from 2010 to 2011. Each check-in record includes a user ID, a location ID, and a timestamp, where each location has its latitude, longitude and category, and each user is associated with her friends. In addition, we used the API of darksky.net³ to collect the weather information for each $\langle \text{latitude}, \text{longitude}, \text{timestamp} \rangle$ record, including temperature, humidity and cloud cover. To evaluate the performances of the proposed method HeteGeoRankRec⁴, implemented based on LibRec [18], we construct two datasets via extracting the check-in records generated from Los Angeles and San Diego. The detailed statistics of the datasets are shown in Table II.

TABLE II. STATISTICS OF DATASETS

	#Users	#POIs	#Check-ins	Sparsity
Los Angeles	2,026	8,270	51,917	99.83%
San Diego	916	4,919	26,762	99.71%

In order to make the experiments more consistent with real situation, we split training data D_{train} and testing data D_{test} as follows: for each individual user, (a) aggregating her check-ins for each location; (b) sorting the location according to the first time that the user checked in; (c) selecting the earliest 80% to train the model and using the next 20% as testing.

2) *Evaluation Metrics*: We employ two widely used metrics to evaluate the performance of different recommendation methods, namely precision and recall, denoted by Pre@K and Rec@K, where K is the number of recommended POIs. We compute Pre@K and Rec@K as follows:

$$Pre@K = \frac{1}{|D_{test}|} \sum_{u \in D_{test}} \frac{|R_u \cap T_u|}{|R_u|} \quad (19)$$

$$Rec@K = \frac{1}{|D_{test}|} \sum_{u \in D_{test}} \frac{|R_u \cap T_u|}{|T_u|} \quad (20)$$

where R_u represents the Top-K recommendation results for user u , and T_u is a set of POIs visited by user u in D_{test} .

3) *Parameters Settings*: We use the meta paths listed in Table I to calculate the semantic correlativity matrixes and add time and weather context constraints to M_3 and M_5 . We divide the time of day into three slices and the weather indicators into three segments in tertile, and build the semantic correlativity matrixes by the method described in Section IV.A. The parameters of check-in probability are obtained through

learning. In particular, we set the latent feature number f of the matrix factorization model to 10, the geographical influence factor β to 0.8, and the regularization parameter λ to 0.01.

4) *Baseline Methods*: We compare the proposed method with the following baseline methods:

- WRMF [16]: A matrix factorization model for implicit feedback.
- BPRMF [17]: A matrix factorization model which optimizes the ordering of the preference for the observed location and the unobserved location.
- GMF: A matrix factorization model based on that proposed in Section IV.B and check-in matrix (correlativity matrix generated from UP meta path) directly for recommendation.
- USG [2]: A model combining user preferences, social relationships, and geographical influence with collaborative filtering.
- RankGeoMF [3]: A matrix factorization model based on ranking and geographical influence for POI recommendation.
- ASMF [10]: A model which learns a set of user's potential locations from her three types of friends, and then incorporates them into matrix factorization.

B. Experimental Result

1) *Performance Comparison*: The comparisons between the HeteGeoRankRec and other methods in terms of Pre@K and Rec@K is shown in Fig. 4. Both WRMF and BPRMF are recommendation methods for implicit feedback data. Due to the severe data sparsity problem of LBSN, these two methods do not perform well. However, we observe GMF improves WRMF by 55.7% and 19.5% in terms of Pre@5 on Los Angeles and San Diego datasets, respectively, due to the incorporation of geographical influence. Besides, USG exhibits better performance than RankGeoMF on both datasets. One possible reason is that USG integrates geographical, social information and user preference, while RankGeoMF only uses geographical information. Most importantly, on average, the proposed HeteGeoRankRec outperforms its competitors WRMF, BPRMF, GMF, USG, RankGeoMF and ASMF, in terms of Pre@5, by 81.1%, 70.5%, 31.4%, 27.2%, 49.5% and 11.4% respectively.

2) *Context Influence*: The performance comparisons of HeteGeoRankRec with different contexts are shown in Fig. 5, which indicate the limited benefit when it only introduces one type of context information. However, combining the time and weather context will greatly improve the Pre@K and Rec@K on both datasets. Therefore, it can be easily concluded that considering various contexts to mine the user's behavior from multiple dimensions makes the model more accurate.

² <https://dropbox.com/s/pa1mni3h8qdkdb/Foursquare.zip?dl=0>

³ <https://darksky.net/dev>

⁴ <https://github.com/Skyexu/HeteGeoRankRec>

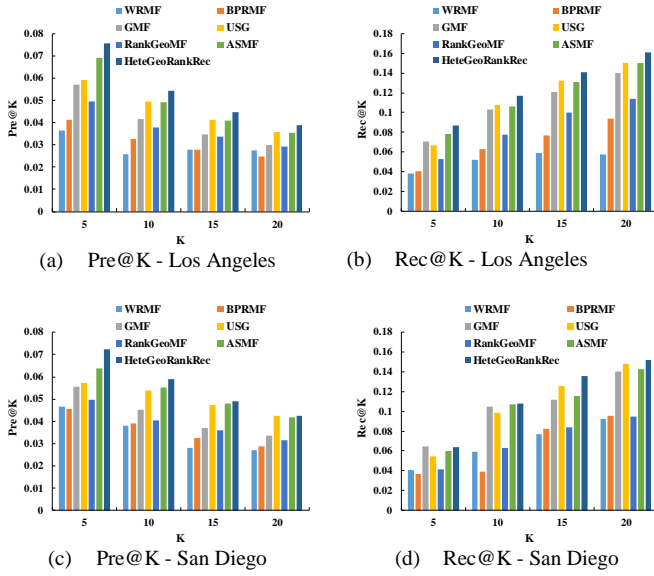


Figure 4. Performance comparisons of different methods.

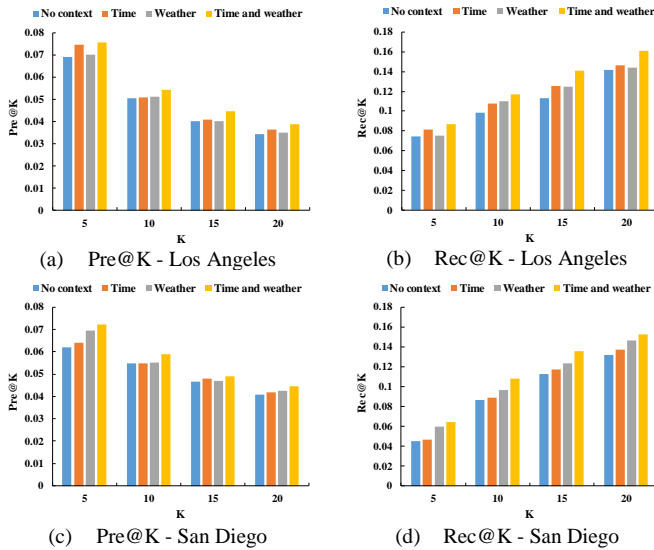


Figure 5. Performance comparisons of different contexts.

VI. CONCLUSION

In this paper, we propose a POI recommendation method called HeteGeoRankRec based on the contextual behavioral semantics. It employs meta paths to represent the complex semantic relationship of user behavior, and combines social relationships, location categories, time and weather contexts, and geographical distance to mine the fine-grained user behavioral characteristics. In the future, we will further study the following issues: (a) deeply explore the influence factors of user behavior in LBSN; (b) express more information on the LBSN heterogeneous information network; and (c) study POI recommendation at specific contexts (e.g. time, weather).

REFERENCES

- [1] Y. Liu, T. Pham, G. Cong, Q. Yuan, "An experimental evaluation of point-of-interest recommendation in location-based social networks," in *Proceedings of the VLDB Endowment*, vol 10, no 10, 2017, pp. 1010-1021.
- [2] M. Ye, P. Yin, W. C. Lee, D. Lee, "Exploiting geographical influence for collaborative point-of-interest recommendation," in *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, 2011, pp. 325-334.
- [3] X. Li, G. Cong, X. Li, T. Pham, S. Krishnaswamy, "Rank-geofm: A ranking based geographical factorization method for point of interest recommendation," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, 2015, pp. 433-442.
- [4] H. Gao, J. Tang, H. Liu, "Exploring social-historical ties on location-based social networks," in *Proceedings of the 6th International AAAI Conference on Weblogs and Social Media*, 2012.
- [5] H. Gao, J. Tang, X. Hu, H. Liu, "Exploring temporal effects for location recommendation on location-based social networks," in *Proceedings of the 7th ACM Conference on Recommender Systems*, ACM, 2013, pp. 93-100.
- [6] C. Trattner, A. Oberegger, L. Eberhard, D. Parra, L. Marinho, "Understanding the Impact of Weather for POI Recommendations," in *RecTour@ RecSys*, 2016, pp. 16-23.
- [7] D. Yu, Y. Wu, C. Liu, X. Sun, "Collective POI Querying Based on Multiple Keywords and User Preference," in *Proceedings of the 24th International Conference on Database Systems for Advanced Applications*, Springer, 2019, pp. 609-625.
- [8] H. Li, Y. Ge, D. Lian, H. Liu, "Learning User's Intrinsic and Extrinsic Interests for Point-of-Interest Recommendation: A Unified Approach," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 2017, pp. 19-25.
- [9] H. Wang, H. Shen, W. Ouyang, X. Cheng, "Exploiting POI-Specific Geographical Influence for Point-of-Interest Recommendation," in *Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI)*, ACM, 2018, pp. 3877-3883.
- [10] H. Li, Y. Ge, R. Hong, H. Zhu, "Point-of-interest recommendations: Learning potential check-ins from friends," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, ACM, 2016, pp. 975-984.
- [11] S. Xing, Q. Wang, X. Zhao, T. Li, "Content-aware point-of-interest recommendation based on convolutional neural network," *Applied Intelligence*, vol 49, 2019, pp. 858-871.
- [12] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, J. Han, "Personalized entity recommendation: A heterogeneous information network approach," in *Proceedings of the 7th ACM international conference on Web search and data mining*, ACM, 2014, pp. 283-292.
- [13] H. Zhao, Q. Yao, J. Li, Y. Song, D. L. Lee, "Meta-graph based recommendation fusion over heterogeneous information networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2017, pp. 635-644.
- [14] Z. S. Wang, J. F. Juang, W. G. Teng, "Predicting poi visits with a heterogeneous information network," *Technologies and Applications of Artificial Intelligence (TAAI)*, IEEE, 2015, pp. 388-395.
- [15] Y. Sun, J. Han, X. Yan, P. S. Yu, T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," in *Proceedings of the VLDB Endowment* 4, 2011, pp. 992-1003.
- [16] Y. Hu, Y. Koren, C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Proceedings of the 8th IEEE International Conference on Data Mining*, IEEE, 2008, pp. 263-272.
- [17] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-Thieme, "BPR: Bayesian personalized ranking from implicit feedback," in *Proceedings of the 25th conference on uncertainty in artificial intelligence*, AUAI Press, 2009, pp. 452-461.
- [18] G. Guo, J. Zhang, Z. Sun and N. Yorke-Smith, "LibRec: A Java Library for Recommender Systems," in *Posters, Demos, Late-breaking Results and Workshop Proceedings of the 23rd Conference on User Modelling, Adaptation and Personalization(UMAP)*, 2015.

A Deep Learning Model Based on Sparse Matrix for Point-of-Interest Recommendation

Jun Zeng

School of Big Data & Software Engineering
Chongqing University
Chongqing, China
zengjun@cqu.edu.cn

Haoran Tang

School of Big Data & Software Engineering
Chongqing University
Chongqing, China
tanghaoran@cqu.edu.cn

Yinghua Li

School of Big Data & Software Engineering
Chongqing University
Chongqing, China
yinghuali@cqu.edu.cn

Xin He

School of Big Data & Software Engineering
Chongqing University
Chongqing, China
hexin@cqu.edu.cn

Abstract—Point-of-interest (POI) recommendation that consists of location-based social networks (LBSNs) and provides personal services for users has become an important part in the field of recommendation system. Due to the sparseness of user check-in matrix, POI recommendation faces great challenges. However, most researches just consider of spatial and temporal impact on recommendation and do not solve the problem of sparsity. This paper proposes a POI recommendation model called RBMNMF which is based on sparse matrix of user check-ins. Firstly, by stacking restricted Boltzmann machines (RBM), the potential relationship between users and POIs is learned and multiple user-POI matrices are extracted. Second, fill the original sparse matrix by using non-negative matrix factorization (NMF). Finally, fuse those prediction matrices to generate final POI recommendation for users, which is benefit for solving the problem of sparsity effectively. Experiments on real-world data set prove that the model we propose has a better accuracy than traditional algorithms.

Keywords—Point-of-Interest Recommendation, Social Network, Restricted Boltzmann Machine, Non-Negative Matrix Factorization, Hybrid Mode

I. INTRODUCTION

Mobile internet technology has developed at a high speed which makes location-based social networks (LBSNs) become popular such as Foursquare, Gowalla, Yelp and Facebook. Compared with traditional social networks, LBSNs allow users share locations with their friends by check-in on POIs (such as cinemas, amusement parks and restaurants). The number of POIs has increased faster at present, so it's necessary to recommend satisfying POIs to users for saving choice time and improving their experience of life in city. However, mining locations among a large number of POIs that user may have interest in is a great challenge.

Different from traditional recommendation system, the challenge of POI recommendation is bigger because the user-POI check-in matrix is a high-dimension sparse matrix, which makes analysis of user-POI matrix more difficult. From some popular data sets such as Foursquare and Gowalla, we observe that sometimes a user just visited a few locations, leading to unreasonable recommendations under the whole location space. Therefore, data sparsity has become a critical problem for POIs recommendation, which is needed to be emphasized and solved. The classical collaborative filtering (CF) algorithm calculates the similarity between users or recommend locations based on check-ins. But faced with the high-dimension sparse matrix of user check-in data, the similarity calculation of users could not be achieved and CF does not solve the problem of data sparsity.

For overcoming the sparsity, this paper proposes a deep learning model based on sparse user check-in matrix for POI recommendation, which constructs bidirectional analysis of users and POIs and aims to recommend POIs to users that meet users' preferences. The main contributions of this paper are as follows.

- Propose a deep learning model called RBMNMF for POI recommendation that is based on sparse matrix of user check-ins.
- Construct strong correlation values between users and POIs to observe and analyze users' preferences for POIs by calculating bidirectional scores between users and POIs.
- Combine stacking restricted Boltzmann machines with non-negative matrix factorization to solve the problem of data sparsity fundamentally.
- Experiments on real-world data set proves that RBMNMF we propose outperforms other traditional algorithms in terms of data sparsity.

DOI reference number: 10.18293/SEKE2019-156

This paper consists of five sections. Section 2 introduces some related algorithms of POI recommendation in LBSNs. Section 3 explicates the RBMNMf model we propose for sparse matrix of user check-ins. Section 4 shows experiments we have and the performance of our model. Section 5 summarizes this paper and discusses the future work.

II. RELETED WORK

At present, location-based POI recommendation in social networks has attracted wide attention from researchers who comes from different fields. Quan et al. [1] considered that the time factor plays an important role in POI recommendation because users visit different locations in different time periods and proposed a collaborative recommendation algorithm combining time factor. References [2-5] regarded users' mobile trajectories as key information when recommending and mined those trajectories to recommend POIs. Ramesh [6] proposed a fusion algorithm that integrated time, space and social relationships into a unified framework for POI recommendation. There have been many achievements for POI recommendation so far, which can be divided into the following three aspects.

- Collaborative filtering (CF): Although CF is the most famous recommendation algorithm, accurate recognition for similar user is a great challenge to CF because of the sparse matrix of user check-ins. Meanwhile, CF ignores lots of user information that is helpful for recommending. Shu et al. [7] proposed a collaborative filtering recommendation algorithm based on topic model to extract user preference of topics (such as culture, history, landscape and so on) information and recommended comprehensive POIs for users. Yang et al. [8] designed a general semi-supervised learning framework based on context information and reduced data sparsity by smoothing adjacent users.
- POI recommendation based on mobile trajectory: User's daily trajectory of movement is an important behavior pattern. Ya [9] extracted semantic information from user GPS trajectory to recommend POIs for users based on time, space and popularity. Zheng et al. [10] proposed a possible path algorithm for uncertain trajectory based on historical trajectory to reduce the uncertainty of user's trajectory. Thus, it is obvious that trajectory plays an important role for POI recommendation.
- POI recommendation based on geographical impact: Distance of a location has a great impact on user's preference for POI. For example, people will not tend to choose location that is far away from people's current locations. Therefore, recommendation system could filter out distant locations [11-13]. Ying et al. [14] considered user's social intention, preference, location popularity and other factors to calculate prior probability of POIs for users.

All algorithms mentioned above recommend POIs for users only by considering additional information of users' sparse check-in matrix and do not solve the problem of data sparsity effectively. However, some researchers focus on the technologies that aims to solve the problem of data sparsity such as matrix factorization [15, 16]. Yildirim [17] use PageRank

algorithm to improve cosine similarity method and alleviated data sparsity. Moreover, Ruslan [18] used restricted Boltzmann machine to handle large-scale data that achieved good performance.

In this paper, we propose a deep learning model called RBMNMf based on sparse matrix of user check-ins for POI recommendation. RBMNMf combines the neural network with non-negative matrix factorization and calculates strong correlation values between users and POIs from bidirectional way, which makes the final prediction matrix smoother. Moreover, RBMNMf are able to display users' preferences for POIs intuitively and solve the problem of data sparsity effectively. Moreover, RBMNMf has a higher accuracy when recommending and outperform some single sparse matrix filling algorithms.

III. HYBRID MODEL BASED ON SPARSE MATRIX OF USER CHECK-INS

A. Definitions

This section defines the sparse user check-in matrix, elaborates on issues of our research and presents the framework of the model we propose. User historical check-ins based on LBSNs includes user ID, location ID, longitude and latitude, check-in time and so on. For simplicity, table 1 shows the meanings of all the symbols in this paper.

TABLE I. THE MEANINGS OF ALL SYMBOLS

Symbol	Meaning
u	user ID
l	location ID
$s_{u,l}$	User score of location
$s_{l,u}$	Score of correlation between user and location
x_u	User vector
x_l	Location vector
U	Set of all users
L	Set of all locations
$M_{u,l}$	Sparse user-location matrix based on user check-ins
$M_{l,u}$	Sparse location-user matrix based on user check-ins

Definition 1: (POI). If user u has a check-in at a location l , then location l is regarded as a point-of-interest (POI). For example, l_j and l_k are two different POIs that user u has visited.

Definition 2: (Sparse User Check-in Matrix). From the LBSNs data set, construct the check-in matrix $M_{u,l}$ and s_{u,l_j} denotes the number of check-ins of user u_i at the location l_j , where $u_i \in U$, $l_j \in L$.

Definition 3: (Sparse User Check-in One-Zero Matrix). If the values of elements in check-in matrix $M_{u,l}$ are greater than 0, then set those values to 1, otherwise 0. Then we obtain the sparse one-zero matrix ${}^0_1 M_{u,l}$.

Definition 4: (Location-User Correlation One-Zero Matrix). Transpose the sparse user check-in one-zero matrix ${}^0_1 M_{u,l}$ to location-user correlation one-zero matrix ${}^0_1 M_{l,u}$.

Where ${}^0_1 M_{l,u} = ({}^0_1 M_{u,l})^T$ and $s_{l,u} \in {}^0_1 M_{l,u}$ which is the value in the matrix ${}^0_1 M_{l,u}$. s_{l_j,u_i} denotes the score of the correlation for a location l_j on user u_i .

B. Model framework

In order to solve the sparsity problem of user check-in data, this paper proposes a hybrid model called RBMNMF that combines deep learning models with non-negative matrix factorization. The hybrid model considers both user-location and

location-user bidirectional information. The model framework is shown in Fig.1.

Sparse user check-in matrix is inputted into stacking RBMs, NMF model and single RBM respectively, and then fuse result of each part to produce a recommendation list after each part calculates its own prediction matrix. RBMNMF model is composed of three parts mentioned above and each part is independent. Moreover, RBMNMF can effectively solve the problem of data sparsity.

C. Restricted Boltzmann Machine Based on Sparse Matrix

Restricted Boltzmann machine (RBM) is an undirected graph probability model with one layer of visible variables and one layer of latent variables, which is shown in Fig.2.

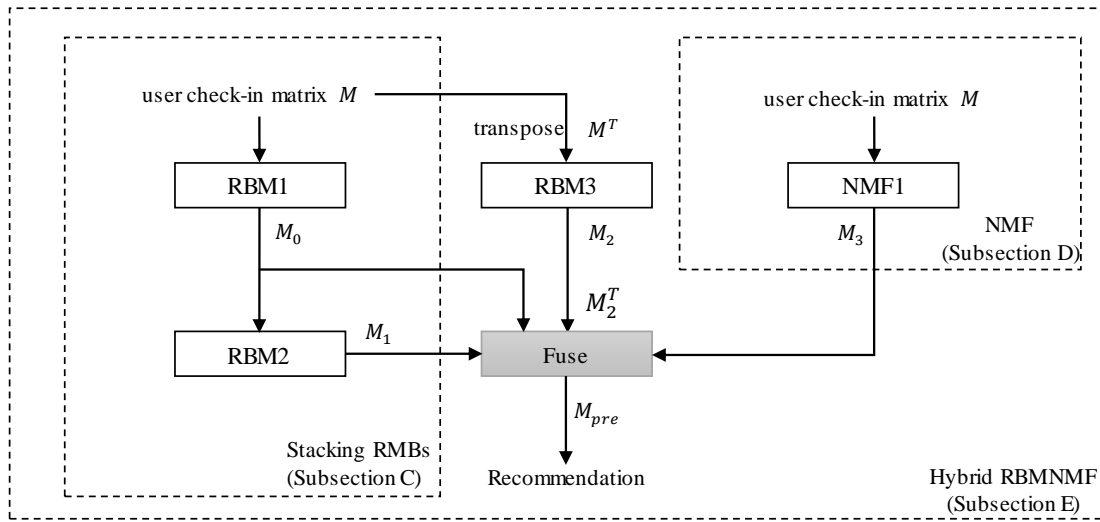


Fig.1 The main architecture of our proposed mode

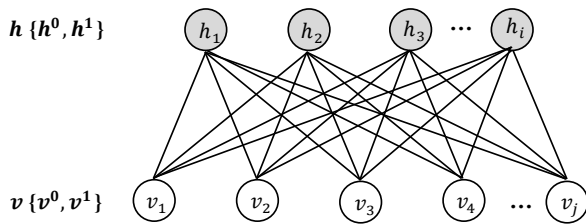


Fig.2 Structure of RBM

RBM model has been widely used in binary data distribution. RBM is defined as binary random vector $\mathbf{v} \in \{0,1\}^d$ that is an energy-based model. The energy function is as (1), where \mathbf{a} and \mathbf{b} are offset vectors of hidden layer and visible layer respectively. Other symbols will be explained latter.

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}) &= -\mathbf{v}^T \mathbf{W} \mathbf{h} - \mathbf{b}^T \mathbf{v} - \mathbf{a}^T \mathbf{h} \\ &= -\sum_i \sum_j w_{ij} v_i h_j - \sum_i b_i v_i - \sum_j a_j h_j \end{aligned} \quad (1)$$

The specific training process of our RBM model is as follows, where \mathbf{v}^0 and \mathbf{v}^1 are visible-layer vectors (\mathbf{v}^0 is also a binary check-in vector at the beginning), \mathbf{h}^0 and \mathbf{h}^1 are hidden-layer vectors, \mathbf{W} denotes the parametric matrix and β is learning rate.

Calculate \mathbf{v}^0 :

Input a user vector \mathbf{x} and the visible-layer $\mathbf{v}^0 = \mathbf{x}$

Calculate \mathbf{h}^0 :

Calculating the Opening Probability of Hidden-Layer Neurons:

$$P(\mathbf{h}_j^0 = 1 | \mathbf{v}^0) = \sigma \left(\sum_i w_{ij} \mathbf{v}_i^0 + a_j \right) \quad (2)$$

The hidden-layer vector \mathbf{h}^0 is obtained by random values filtered from 0 to 1

Calculate \mathbf{v}^1 :

\mathbf{h}^0 reconstructs visible layer:

$$P(\mathbf{v}_i^1 = 1 | \mathbf{h}^0) = \sigma \left(\sum_j w_{ij} \mathbf{h}_j^0 + b_i \right) \quad (3)$$

The visible-layer vector \mathbf{v}^1 is obtained by random values filtered from 0 to 1

Calculate \mathbf{h}^1 :

\mathbf{v}^1 reconstructs hidden layer:

$$P(\mathbf{h}_j^0 = 1 | \mathbf{v}^1) = \sigma \left(\sum_i w_{ij} \mathbf{v}_i^1 + a_j \right) \quad (4)$$

The hidden vector \mathbf{h}^1 is obtained by random values filtered from 0 to 1

Update Parametric Matrix:

$$\mathbf{W} + \beta \left\{ (\mathbf{h}^0)^T \times \mathbf{v}^0 - (\mathbf{h}^1)^T \times \mathbf{v}^1 \right\} \rightarrow \mathbf{W} \quad (5)$$

$$\mathbf{b} + \beta (\mathbf{v}^0 - \mathbf{v}^1) \rightarrow \mathbf{b} \quad (6)$$

$$\mathbf{a} + \beta (\mathbf{h}^0 - \mathbf{h}^1) \rightarrow \mathbf{a} \quad (7)$$

In order to prevent over-fitting and make our model smoother, data transformation and stacking RBMs are used to train data set, which is shown in Fig.3.

Firstly, input ${}^0_1 M_{u,l}$ that is the original user check-in matrix to stacking restricted Boltzmann machines model for sparse data training and filling, and then predict the location score M_0 and M_1 for users. The combination of two RBMs will enhance the matrix computation and make prediction more reasonable, which is demonstrated from our experiments.

Secondly, input ${}^0_1 M_{l,u}$ to a single RBM and then predict the user score M_2 for locations. ${}^0_1 M_{l,u}$ is the transposition of ${}^0_1 M_{u,l}$. The reason why we take this measure is that most existing works only consider of users' preference for POIs and ignore the POIs attraction to users. Thus, we decide to take transposition into account. The final predictive matrix M'_{pre} of restricted Boltzmann machine model based on sparse matrix is as follows.

$$M'_{pre} = M_0 + M_1 + (M_2)^T \quad (8)$$

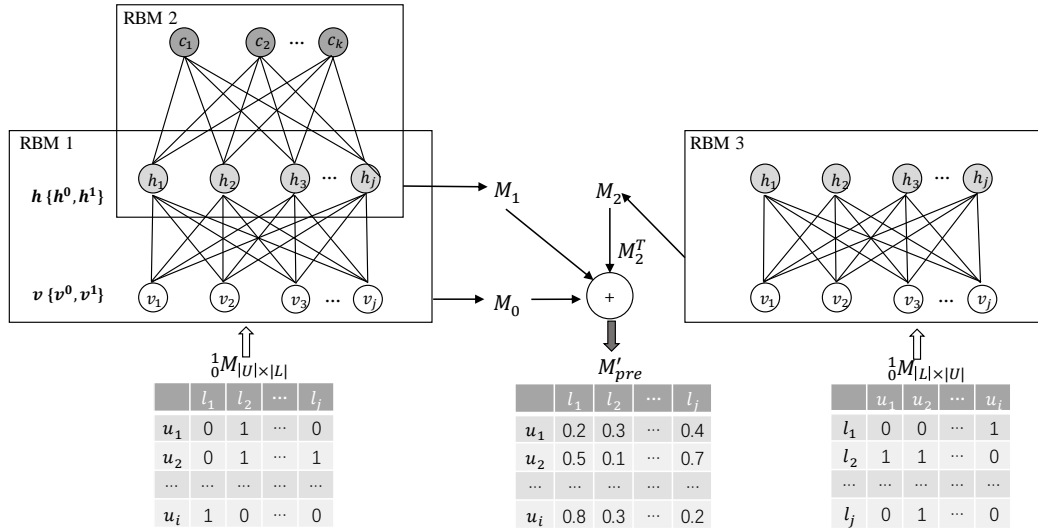


Fig.3 The workflow of stacking RBMs

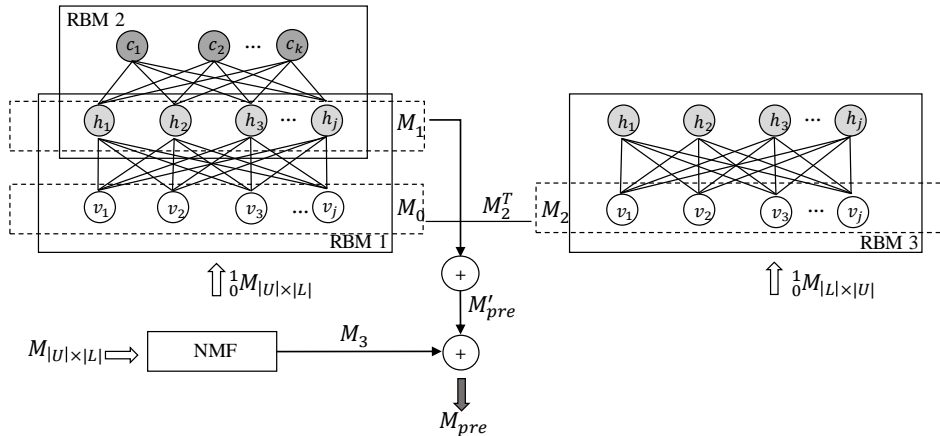


Fig.4 Combination of stacking RBMs and NMF

D. Non-Negative Matrix Factorization Model Based on Sparse Matrix

Values of elements in sparse user check-in matrix are all non-negative. However, the traditional matrix factorization model will get negative values from original matrix, which have no practical significance for user check-in matrix.

Non-negative matrix factorization (NMF) can factorize the user check-in matrix into two non-negative matrices and fill the sparse matrix with positive values. Therefore, it solves the sparse problem of matrix effectively and it's easy to observe the user's preference for locations. NMF is shown in Fig.4. $M_{u,n}$ and $M_{n,l}$ is the result of factorization of $M_{u,l}$. Thereafter, M_3 is the multiplication of them, which is filled with more negative values.

	k_1	k_2	\cdots	k_n			l_1	l_2	\cdots	l_j			l_1	l_2	\cdots	l_j
u_1	0	0.8	\cdots	0	\times	k_1	0	0.5	\cdots	1.9	$=$	u_1	0	1.8	\cdots	0
u_2	2.0	0	\cdots	0		k_2	0	2.3	\cdots	0		u_2	0	1	\cdots	3.8
\cdots	\cdots	\cdots	\cdots	\cdots		\cdots	\cdots	\cdots	\cdots	\cdots		\cdots	\cdots	\cdots	\cdots	
u_i	0	0	\cdots	2.5		k_n	1.2	1	\cdots	0		u_i	3	2.5	\cdots	0
$M_{ U \times N }$						$M_{ N \times L }$						M_3				

Fig.5 The result of factorization of NMF

In $M_{u,l}$, the value of each element represents the times that the user has visited the location. As shown above, using NMF model, predictive matrix M_3 fills the elements of $M_{u,l}$. Each element's value of M_3 denotes the preference of a user for a location.

E. Hybrid RBMNMF Model

In order to make our model smoother, we propose a hybrid model called RBMNMF that fuses the models mentioned in the above sections, as shown in figure 4.

The whole process of RBMNMF is as follows.

1) Input $M_{u,l}$ into two stacking RBMs model for sparse data training and filling. Then predict location score matrices M_0 and M_1 .

2) Input $M_{l,u}$ into a single RBM and predict the location-user correlation score matrix M_2 . Then transpose the M_2 .

3) Factorize $M_{u,l}$ into two non-negative matrices to obtain another location score matrix M_3 for users.

4) Predict the final recommendation matrix as follows.

$$M_{pre} = (M_0 + M_1 + (M_2)^T) + M_3 \quad (9)$$

5) Recommend Top-N POIs to users according to M_{pre} .

IV. EXPERIMENTS

A. Datasets

Foursquare is a social networking site that records a large number of geographic information of users' current locations. This paper uses the data set provided by Quan et al. [1] which contains 342850 check-ins from August 2010 to July 2011 in Singapore. Yuan [1] deletes users whose check-ins are less than 5 and locations checked by less than 5 users. After processing, the foursquare data set contains 2321 users, 5596 POIs and 194108 check-ins. For each user, choose 12.5% of the user's POIs for tuning parameters and another 25% of the user's POIs for testing data randomly. The left POIs are used for training. The formats of each check-in is user ID, location ID, longitude and latitude, check-in time and time ID respectively.

B. Evaluation Metric

We use *Precision@N* and *Recall@N* to evaluate our model, which are as follows.

$$precision @ N = \frac{|R(u) \cap T(u)|}{R(u)} \quad (10)$$

$$Recall @ N = \frac{|R(u) \cap T(u)|}{T(u)} \quad (11)$$

$R(u)$ denotes the list of POIs recommended to user u and $T(u)$ denotes the list of POIs that are actually checked in by user u in the test data. $|R(u)|$ is the total number of POIs in $R(u)$ and $|T(u)|$ is the total number of POIs in $T(u)$. The final precision and recall are the average of all users.

C. Experimental Result

In order to verify the recommendation effectiveness of our RBMNMF model, we compare RBMNMF with single sparse matrix filling algorithms in the same training data and test data. Those algorithms are as table 2 shows. The core of this paper is to integrate non-negative matrix factorization and stacking RBMs. So we focus on comparing our proposed algorithm with single NMF or single RBM but CF is a popular algorithm that we should also consider.

TABLE II. COMPARISON ALGORITHMS

Algorithm	Description
CF	Collaborative filtering (CF) is the most popular and classical recommendation algorithm
RBM	Single restricted Boltzmann machine based on sparse matrix of user check-ins
NMF	Single non-negative matrix factorization based on sparse matrix of user check-ins
RBMNMF	The hybrid model we propose

The experimental results are shown in Fig.6 and Fig.7. We recommend N (N = 5, 10, 20, 30) POIs for each user. The performance of CF is the worst among all algorithms and possibly due to the data set, the precision of CF does not change significantly. No matter what the value of N is, the precision

and recall of RBMNMF are generally better than other single algorithms. Hence, even though there is a lack of user information, RBMNMF we propose are able to fill the sparse matrix well and have an excellent performance when recommending. The results of experiments demonstrate RBMNMF could be used into improving traditional algorithms.

However, there is a limitation in RBMNMF. When stacking RBMs process check-in data, we adopt the behavior of check-in to represent the user's preference for a location by 0 or 1. That treats all locations equally that a user has visited even though they have different numbers of check-ins.

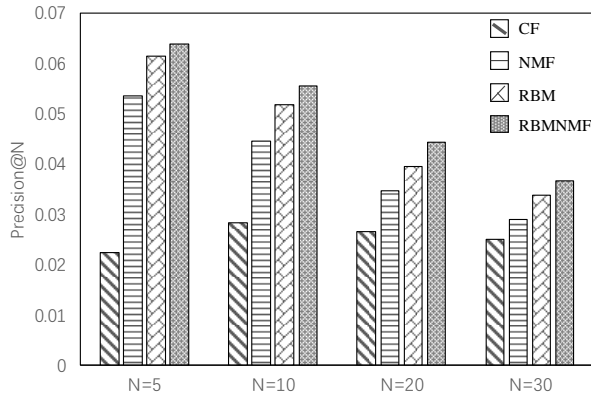


Figure 6. Precision with different N

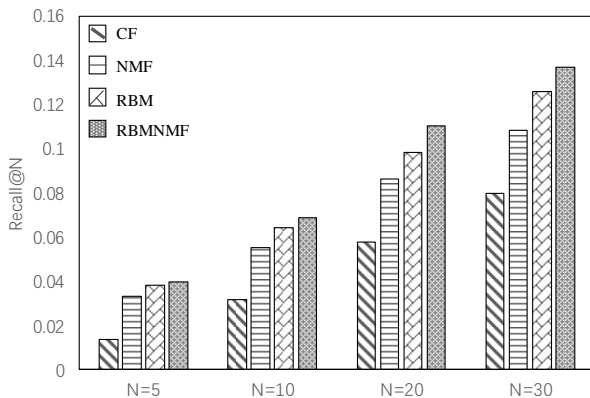


Figure 7. Recall with different N

V. CONCLUSION

For the sake of overcoming the sparsity problem of user check-in matrix, we propose a deep learning model called RBMNMF that combines stacking restricted Boltzmann machines with non-negative matrix factorization to make the prediction model smoother. RBMNMF fills the user check-in matrix and alleviates the problem of data sparsity effectively. Therefore, we could observe each user's preference value for each POI intuitively and then recommend satisfying POIs to users. In the future, more information will be taken into account such as time, geographical effect and location popularity to improve our model.

ACKNOWLEDGMENT

This research is supported by the National Natural Science Foundation of China (Grant No. 61502062, Grant No. 61672117 and Grant No. 61602070).

REFERENCES

- [1] Yuan, Quan , et al. "Time-aware point-of-interest recommendation." *International Acm Sigir Conference on Research & Development in Information Retrieval ACM*, pages 363-372, 2013.
- [2] Yu, Zheng, et al. "Recommending friends and locations based on individual location history." *Acm Transactions on the Web*, pages 1-44, 2011.
- [3] Yu, Zheng, and X. Xing. "Learning Location Correlation from GPS Trajectories." *Eleventh International Conference on Mobile Data Management*, pages 27-32, 2010.
- [4] Zheng, Yu , and X. Xie . "Learning travel recommendations from user-generated GPS traces." *ACM Transactions on Intelligent Systems and Technology*, pages 1-29, 2011.
- [5] Ye, Mao , et al. "Exploiting geographical influence for collaborative point-of-interest recommendation." *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 325-334, 2011.
- [6] Baral, Ramesh , and T. Li . "MAPS: A Multi Aspect Personalized POI Recommender System." *Acm Conference on Recommender Systems ACM*, pages 281-284, 2016.
- [7] Jiang, Shuhui , et al. "Author topic model-based collaborative filtering for personalized POI recommendations." *IEEE Transactions on Multimedia*, pages 907-918, 2015.
- [8] Yang, Carl , et al. "Bridging collaborative filtering and semi-supervised learning: A neural approach for POI recommendation." *Acm Sigkdd International Conference on Knowledge Discovery & Data Mining ACM*, pages 1245-1254, 2017.
- [9] Liu, Yaqiong , and H. S. Seah . "Points of interest recommendation from GPS trajectories." *International Journal of Geographical Information Systems*, pages 953-979, 2015.
- [10] Zheng, Kai , et al. "Reducing Uncertainty of Low-Sampling-Rate Trajectories." *2012 IEEE 28th International Conference on Data Engineering IEEE Computer Society*, pages 1144-1155, 2012.
- [11] Bao, Jie , Y. Zheng , and M. F. Mokbel . "Location-based and preference-aware recommendation using sparse geo-social networking data." *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 199-208, 2012.
- [12] Ferenc, Gregory , M. Ye , and W. C. Lee . "Location recommendation for out-of-town users in location-based social networks." *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management ACM*, pages 721-726, 2013.
- [13] Wang, Hao , M. Terrovitis , and N. Mamoulis . "Location recommendation in location-based social networks using user check-in data." *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 364-373, 2013.
- [14] Ying, Jia Ching , et al. "Mining User Check-In Behavior with a Random Walk for Urban Point-of-Interest Recommendations." *ACM Transactions on Intelligent Systems and Technology*, pages 1-26, 2014.
- [15] Ma, Hao , et al. "SoRec: Social recommendation using probabilistic matrix factorization." *ACM*, pages 931-940, 2008.
- [16] Lian, Defu , et al. "GeoMF : Joint Geographical Modeling and Matrix Factorization for Point-of-Interest Recommendation." *Acm Sigkdd International Conference on Knowledge Discovery & Data Mining ACM*, pages 831-840, 2014.
- [17] Yildirim, Hilmi , and M. S. Krishnamoorthy . "A random walk method for alleviating the sparsity problem in collaborative filtering." *Acm Conference on Recommender Systems ACM*, pages 131-138, 2008.
- [18] Salakhutdinov, Ruslan , A. Mnih , and G. Hinton . "[ACM Press the 24th international conference - Corvallis, Oregon (2007.06.20-2007.06.24)] Proceedings of the 24th international conference on Machine learning - ICML '07 - Restricted Boltzmann machines for collaborative filtering." (2007):791-798.

Improving Code Generation From Descriptive Text By Combining Deep Learning and Syntax Rules

Xiangru Tang^{a,b}, Zhihao Wang^c, Jiyang Qi^c, Zengyang Li^{a,b,*}

^aSchool of Computer Science, Central China Normal University, Wuhan, China

^bHubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning, Central China Normal University, Wuhan, China

^cSchool of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China

xrtang@mails.ccn.edu.cn, zhihaowang@hust.edu.cn, jyqi@hust.edu.cn, zengyangli@mail.ccn.edu.cn

Abstract—Code generation is a model-driven engineering approach that enables developers to generate source code automatically and achieves extremely high development productivity. Specifically, generating code from a descriptive text reduces the time and expense of software development significantly. However, the performance of existing methods is not satisfying, since they are either of low accuracy (lack of specifics of the generated code) or too complicated (lack of efficiency in training). In this work, we proposed three novel methods by combining neural architectures and syntax rules, aiming at explicitly capturing the syntactical characteristics of target code. First, we proposed three models based on the Combination of Deep learning and Syntax rules (CDS models). Then, we evaluated CDS models with BLEU metric by comparing our models with existing methods. The results show that our models outperform existing methods for the challenging code generation task. Finally, we conducted a comparative study between the three CDS models. With further analysis we provided advice on the choice of neural architectures by considering both task accuracy and efficiency. Experimental results show that (1) there is a trade-off between speed and accuracy of the model, and (2) one of our CDS models (i.e., the CDS-POOLING model) outperforms other existing methods for the challenging code generation task.

Index Terms—code generation, neural network, abstract syntax tree, encoder-decoder

I. INTRODUCTION

Code generation is a process of generating source code from sentences that describe code functionality [1]–[3]. Firstly, generating code automatically with given descriptive sentences as constraints is significantly faster than writing the code manually. Moreover, the generated code works in an expected way and is more maintainable and extendable. In contrast, with code written manually, different developers tend to use different styles, which may lead to software errors. Thus, code generation is of great significance throughout the software development lifecycle.

However, code generation is challenging because its output must abide by the syntax rules strictly, and the arithmetic speed



Fig. 1. An example of Hearth Stone dataset

for the purpose of practicability should also be considered. Considerable effort has been invested in code generation, which results in several types of code generation approaches. However, they often fail to generate executable code correctly because they hardly capture complicated code structures.

Recently, deep learning approaches based on neural networks have shown significant performance improvement on many artificial intelligence tasks. Public datasets speed up the development of the code generation area. One of high-quality datasets is Hearth Stone¹, which aims to generate Python code based on Hearth Stone card description. An example is shown in Fig 1, in which a piece of code lies along the bottom. Each card is identified by ten attributes (e.g., name and attack) and has a text box to describe the effect of the card.

In this work, we proposed three models based on the Combination of Deep learning and Syntax rules (CDS models). This work involves some neural architectures (e.g, self-attention network, CNN) which have already resulted in significant progress in the field of natural language processing. Meanwhile, considering the essential difference between

* Corresponding author.

This work is partially supported by the National Natural Science Foundation of China (NSFC) under the Grant Nos. 61702377 and 61773175.
DOI reference number: 10.18293/SEKE2019-170

¹ available at <https://github.com/deepmind/card2code>

code and natural language, our work also takes syntax rules into account. Specifically, we represented the code in a tree structure. In addition, considering the poor performance of Recurrent Neural Network in existing methods [1]–[4], we adopted the pooling operation and self-attention mechanism in code generation tasks. Moreover, we explored the strengths and weaknesses of our CDS models (i.e., CDS-POOLING, CDS-CNN, and CDS-SAN models).

The key contributions of our study are described as follows:

- a) We proposed three novel code generation frameworks, based on the Combination of Deep learning and Syntax rules (CDS models). To the best of our knowledge, we are the first to use Transformer model (self-attention network) and pooling operation in code generation. And our high-performance models lead to a significant improvement of BLEU score.
- b) We conducted a comparative study between three models we proposed. We also thought that researchers need a comprehensive analysis of the task and should adopt simple and effective networks, when using deep learning methods.

II. RELATED WORK

Generating code in software engineering has a long history. Some early works focus on domain specific languages [5], [6]. For general-purpose code generation, such as [7], which tries to generate the code for parsing input documents. It was presented that data driven methods instead of manual methods are used in manufacturing model, and code could be generated automatically.

There are some early works using syntax rules only. (1) Parser Generation approach: some tools such as template engine [8] were used to automatically generate parser, but it is too complicated and cannot cover every scenario. (2) Model Driven approach: an entire application or just its skeleton is generated. (3) Database-related approach: usually the programmer defines a database schema, from which entire CRUD (Create, Retrieve, Update, and Delete) operations or just the code to handle the database can be generated. (4) Metaprogramming approach: some researchers developed a new language which could manipulate another piece of source code; it means that the source code is just another data structure that can be manipulated [9]. (5) Retrieval-based approach: some researchers leveraged subtree retrieval mechanism, which can explicitly output existing code examples. [10].

Recently, neural networks are introduced to code generation. Encoder-Decoder architecture has shown good performance in practical applications, such as machine translation, dialogue system, and image captioning. The encoder processes an input sequence $x = (x_1, \dots, x_m)$ of m elements and returns state representations $z = (z_1, \dots, z_m)$. The decoder takes z and generates the output sequence $y = (y_1, \dots, y_n)$ left to right. To generate output y_{i+1} , the decoder computes a new hidden state h_{i+1} based on the previous state h_i , an embedding g_i of the previous target language word y_i , as well as a conditional input c_i derived from the encoder output z . Based on this generic

formulation, various encoder-decoder architectures have been proposed.

Recent works on neural machine translation mostly base on sequence-to-sequence models are worth referring for us. [11], [12] and many works following adopt attention based models to get better performance with longer sentences. [13] uses CNNs to build sequence-to-sequence model which is faster and allows to discover compositional structure. Transformer is proposed in [14] first. Although originally used in translation tasks, self-attentive feed-forward sequence models have been shown to achieve impressive results on many sequence modeling tasks [14]–[16]. For code generation task, some researchers attempted to adopt sequence-to-sequence models [1] or models based on abstract syntax tree [2]–[4], [17] to get valid program. The decoder of neural network models above are all based on RNNs except for [17] which uses CNNs to capture information.

Moreover, some researchers found that much simpler word-embedding-based architectures exhibit impressive performance, compared with more-sophisticated models using RNN or CNN [14]. There are some related works which introduced pooling to some tasks. [18]–[20] show that average pooling can obtain impressive accuracy on both sentence and document-level sentiment analysis, factoid question-answering and text classification tasks with much less training time than competing methods.

III. ARCHITECTURE

We first define the code generation task as below:

Given a descriptive text q , our target is to generate code (e.g., Python code), specifically in an AST a format. In this paper, we start with the syntactic code generation model proposed in [2]. It focuses on generating AST from text, and then converting it to concrete code. Formally, our goal is to find a best generated AST \hat{a} as Eq.(1):

$$\hat{a} = \arg \max_a p(a|q) \quad (1)$$

$$p(a|q) = \prod_{t=1}^T p(y_t|y_{<t}, q) \quad (2)$$

where $y_{<t}$ represents $y_1 \dots y_{t-1}$ and T is the number of total time steps.

Our CDS models can be divided into three dimensions: pooling based, CNN based, and self-attention network based. We train these three models independently, with input of (a) predicted structure of AST, (b) name of variables, and (c) name of functions containing syntax information.

A. CDS-POOLING

Word embeddings can learn a lot from rich unstructured descriptive text, which are widely adopted as building blocks in the area of Natural Language Processing (NLP). Word embeddings can cluster similar words in semantical level by representing each word as a fixed-length vector and encoding the linguistic regularities and patterns implicitly. Thus, our

CDS-POOLING model is closely related to Deep Averaging Network, which demonstrates that the average pooling operation achieves tremendous results for some NLP tasks. Pooling operations capture high level semantic features and low level word characters information, just same as a method of information fusion. Moreover, we explored different pooling operations, rather than only average-pooling.

Average Pooling: Average pooling computes the element-wise average over word-vectors for the descriptive text, which average the value of K dimensions for all word embedding. Thus, Average Pooling is able to obtain a representation z with the same dimension as the embedding itself.

$$z = \frac{1}{L} \sum_{i=1}^L v_i \quad (3)$$

Max Pooling: Max Pooling extracts the most salient features from every word-embedding dimension, by taking the maximum value along each dimension of the word vectors.

$$z = \text{Max-pooling}(v_1, v_2, \dots, v_L) \quad (4)$$

where the j -th component of z is the maximum element in the set v_{1j}, \dots, v_{Lj} , where v_{1j} is, for example, the j -th component of v_1 . With this pooling operation, those words that are unimportant or unrelated to our task will be ignored in the encoding process.

Hierarchical Pooling: Both average- and max- pooling do not take word order into consideration, which could be useful for code generation tasks. Thus, we also proposed a hierarchical pooling layer, where the two abstracted pooling features are concatenated together to represent the sentence embeddings. However, Hierarchical Pooling learns fixed-length representations for the n -grams that appear in the corpus, rather than just capturing their occurrences via count features, which is more suitable for code generation.

For all CDS-POOLING variants above, there is no additional network to extract features. We just apply max-pooling after embedding, and then attention mechanism is used to acquire information between features. Finally, two layers of fully connect neural network and a softmax layer is applied to get the classification result. Thus, CDS-POOLING model quickly captures only intrinsic word-embedding information for code generation. In experiments, we proved that the CDS-POOLING model significantly promote the precision and accelerate the calculation speed.

B. CDS-CNN

The Convolutional Neural Network (CNN) [21] is another strategy extensively employed encode text sequences. Convolution operation can be described as using filters $\mathbf{w} \in \mathbb{R}^{h \times k}$ to capture word-level features. For each kernel F , a convolution operation uses a $D \times K$ slide. First, We applied the embedding mentioned above as input, which is a tensor of $D \times L$ dimensions. L represents the length of descriptive sentence and D is the embedding dimensions. Then, for each step, we summed the weighted value in filters and applied nonlinear activation

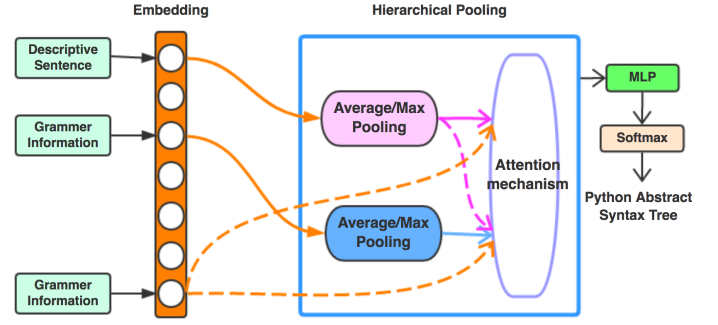


Fig. 2. Simple Word-Embedding-Based Model

operation to obtain the filter. Finally, the filters become a vector which represents the output. In practice, several layers are applied to capture the hierarchical information.

We learned from [22] and applied residual structure in our model. What's more, we also conducted batch normalization [23] to speed up the training process and improve the model performance. Our CDS-CNN model is showed in Fig 3. The embedding input has the shape of $N \times L \times D$. Our designed interact-CNN takes into account the relationship between natural language and semantic structure. Moreover, we use the concatenation operation to mix the information from natural language and AST together. But attention is needed for our operation of transposing the length-dimension and embedding-dimension to fully mix the information. At the final part of interact-CNN, we applied max-pooling to get the hierarchical information. Through the interact-CNN part, the shape of our tensor is reduced to $N \times D$. At the final part of Fig 3, there are multilayer perceptron (MLP) and softmax layer to generate final AST. CNN can be fully trained in parallel to better exploit the floating-point computation capacity of GPU, compared with RNN in the train step.

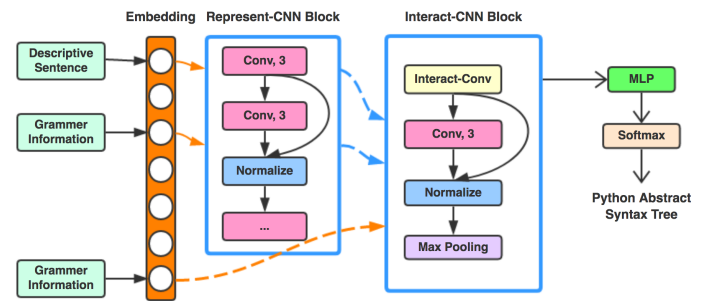


Fig. 3. Encoder-Decoder CNN

C. CDS-SAN

Transformer architecture [14] is based entirely on attention mechanisms and achieves best performance in the neural machine translation. In this work, we adopted the self-attention network architecture, which is a modified version of Transformer for two reasons. Firstly, it could be trained fast for its

parallelizable structure, while traditional RNN model is computational and time consuming due to its recurrent structure. Secondly, it naturally constructs the long-term dependence via the attention mechanism, see Eq 4. It implies that Transformer architecture learns non-local dependencies between tokens regardless of the distance between them.

The self-attention network (SAN) is a special case of the attention mechanism, which models the dependencies between tokens from the same sequence [14]. In our work, the self-attention layer aims to create the embedding representations of the original text input. Specifically, given the text $H = (h_1, h_2, \dots, h_L)$ carrying the semantics of the original review along all time steps from the encoder, self-attention network first yields a weight matrix $A^{enc} = (a_1^{enc}, a_2^{enc}, \dots, a_L^{enc})$, computed as $A^{enc} = \text{softmax}(w_2^{enc} \tanh(w_1^{enc} H^T))$, where w_1^{enc} is a parameter vector and w_2^{enc} is a parameter matrix. *Softmax* function is used to normalize the attention weights.

Since embeddings represent linguistic context information weakly, we use self-attention network to encode input for a better representation and drop out the decode part.

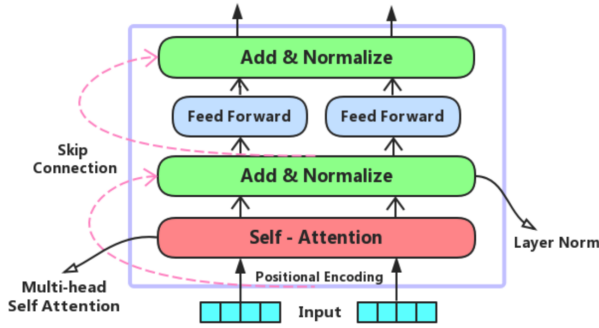


Fig. 4. Self-attention Network

IV. EXPERIMENT

Our experiment aims to answer the following research questions (RQs):

RQ1: How do CDS models perform?

Our work intends to make a comparative study of the differences between our CDS models and existing state-of-the-art methods (e.g., LPN [1], SNM [2], ASN [3], and SEQ2TREE [4] all mentioned above in section II).

RQ2: Can we find a trade-off between training speed and accuracy of the result?

Most models with more expressive composition functions perform well? Speed and accuracy, which matters more in reality? This work includes a rigorous evaluation regarding the added value of sophisticated composition functions.

A. Dataset

We used Hearth Stone dataset introduced by [1] as a practical code-generation application. It has 655 cards in total, 533 cards for training, 66 cards for validating, and 66 cards for testing. Each card is identified by ten attributes (e.g., TYPE

and ATK) and has an functional describe in the text box. Code implementations of these cards implement the game logic and card effects once per turn.

B. Evaluation Metrics

We used bilingual evaluation understudy score (BLEU) [24] as the evaluation metric in our experiment. Although BLEU is developed for translation tasks originally, it can be used to evaluate programs in this work through measuring how close the generated code is to the ground truth code in terms of n -grams.

To obtain the BLEU score, we computed modified n -grams precision (p_n) first. The modified n -grams prediction is computed as follows. All candidate n -grams counts and their corresponding maximum reference counts are collected. In the code generation task, the reference is executable code.

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n\text{-gram} \in C} Count_{clip}(n\text{-gram})}{\sum_{C' \in \{Candidates\}} \sum_{n\text{-gram}' \in C'} Count(n\text{-gram}')} \quad (5)$$

Then, we define BP to penalize the generated program shorter than ground truth. c is the length of generated code and r is the reference code length.

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leq r \end{cases} \quad (6)$$

Finally, we get

$$BLEU-N = BP \cdot \exp\left(\sum_{n=1}^N w_n \log p_n\right) \quad (7)$$

In our experiment, we used $N = 4$ and uniform weights $w_n = 1/N$.

Additionally, we also calculated the ACCURACY score which represents the percentage of 66 test codes that can be executed and absolutely correct.

C. Experiment Hyperparameters

For the CDS-POOLING model, our character embedding has 256 dimensions. We used a dropout rate of 0.7; for the CDS-CNN model, we set the embedding size as 128 to concentrated information and hidden-layer size as 128 to extract features. The dropout rate is 0.5 because CNN model is more complicated than pooling. For the CDS-SAN model, we followed the base Transformer [14], with 4 encoder layers of 128 hidden dimensions, and 8 attention heads per-layer. And the CNN filters followed the specifications of [21]. The dropout rate is also 0.5, same as CDS-CNN. We optimized these models with Adam with default hyperparameters and set batch size as 64. It took us 9 hours to train a pooling model on a NVIDIA Tesla P100. We used a beam search for generating the program, and computed BLEU scores to measure performance on the testing set.

TABLE I
EXPERIMENTAL RESULTS

Model	ACCURACY	BLEU
SEQ2TREE	1.5	53.4
LPN	6.1	67.1
SNM	16.2	75.8
ASN	18.2	77.6
CDS-POOLING	15.6	78.9
CDS-CNN	19.7	77.0
CDS-SAN	16.7	77.8

D. Experimental results

Table I presents the results of our CDS models in code generation, in comparison with existing methods, e.g., LPN [1], SNM [2], ASN [3], and SEQ2TREE [4], which are the state-of-the-art models in the code generation area. The results show that adopting CDS-POOLING to capture the syntax rules yields a better performance with much less training time than other methods. Besides, compared with RNN, CNN is better suited to capturing the structural information of long sentences. Finally, the self-attention network based model also outperforms existing methods.

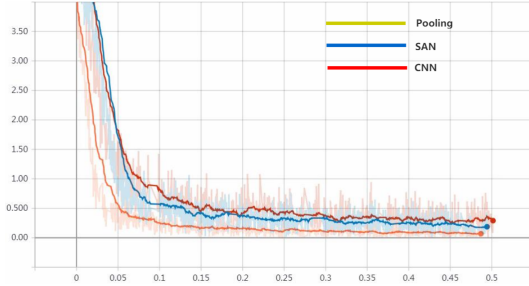


Fig. 5. Loss Function Over Time

E. Result Analysis and Case study

The results demonstrate that CDS-POOLING is particularly better than any other existing methods in both speed and efficiency. As for CDS-SAN, when we made this model more complicated, the ACCURACY score fell but BLEU score rose, which means CDS-SAN can handle the detail better. CDS-CNN model is good at generating more completely correct code.

As mentioned in [3], Hearth Stone contains classes with similar structures, thus the code generation task can predigest into the generation of tree-like AST and what we need to do is to fill in tokens with certain variables and values. Nevertheless, certain errors must occur because that Hearth-Stone's code contains complicated logic, which result in a low accuracy [2]. To see more details, we presented an example of code we generated by CDS-POOLING in table II. It illustrates that our model can handle complicated code syntax effectively. However, our generated code does not absolutely match the reference code, but it is correct in general and can be executed.

To be more specific, the reason why the CDS-POOLING has such an impressive result is that pooling operation builds an

information map to achieve a downsampling-process, which throws away indifferent information. In contrast, the CDS-CNN model tends to generate a structural correct code, which leads to a higher ACCURACY (more absolutely correct codes) but a similar BLEU compared with previous works.

TABLE II
THE COMPARING OF GENERATED CODE AND REFERENCE CODE

Descriptive Text:

```
annoy-o-tron name_end 1 atk_end 2 def_end 2 cost_end -1 dur_end
minion type_end neutral player_cls_end mech race_end
common rarity_end b taunt /b nl b divine shield /b
```

Generated Code:

```
class AnnoyTron(MinionCard) :
    def __init__(self) :
        super().__init__("Annoy-o-Tron", 2,
            CHARACTER_CLASS.ALL, CARD_RARITY.COMMON,
            minion_type = MINION_TYPE.MECH, divine_shield = True)
    def create_minion(self, player) :
        return Minion(1, 2, taunt = True, divine_shield = True)
```

Reference Code:

```
class AnnoyTron(MinionCard) :
    def __init__(self) :
        super().__init__("Annoy-o-Tron", 2,
            CHARACTER_CLASS.ALL, CARD_RARITY.COMMON,
            minion_type = MINION_TYPE.MECH)
    def create_minion(self, player) :
        return Minion(1, 2, divine_shield = True, taunt = True)
```

V. DISCUSSION AND ANALYSIS

RQ1: How do CDS models perform?

The results demonstrate that CDS-POOLING is particularly better than any other existing methods in both speed and efficiency. When we made model more complicated, the ACCURACY result fell, but the BLEU score rose (see CDS-SAN), which means that it can handle the details better. And it clarifies CDS-CNN model is skilled at generating more completely correct code. Results also illustrate that simple pooling operation (CDS-POOLING) is surprisingly effective at representing longer sequence. CDS-POOLING figures over all elements of the tensor, and CDS-POOLING can be fully trained in parallel to better exploit the floating-point computation capacity of GPU, compared with RNN in the train step. In addition, CDS-POOLING is more amenable to optimization because the number of non-linearities unit is fixed, moreover, it not be affected by the text input's length. The max-pooling is working as follows: we can consider the embedding size used as 256 represent 256 kinds of semantics, and the max-pooling operator is to extract the semantics in sentences.

RQ2: Can we find a trade-off between training speed and accuracy of the result?

The experiment demonstrated that CDS-POOLING is the most effective approach outperforming other complicated models. This also tells us that a simple and effective method should be a prevailing concern. Additionally, it helps researchers to avoid the blind use of deep learning algorithms when solving software engineering problems.

Considering the nature of this problem, more sophisticated models reveal outstanding results but are excessively computationally expensive, because they need to optimize thousands of parameters, e.g., RNN or CNN. On the contrary, maybe some simpler models can be robust, which only compute the sentence embedding by simply adding or averaging operation over the word embedding, just as our CDS-POOLING. But that also means, such a simple pooling operation does not take word-order information into account. However, pooling operation has the advantage of having significantly fewer parameters, which means it can train much faster and obtain a equally good precision, comparing to RNN or CNN. Thus, there is a trade-off between training speed and efficiency.

VI. CONCLUSION AND FUTURE WORK

In this work, we are the first to introduce pooling operation, fully convolutional model, and self-attention network for code generation tasks to the best of our knowledge. Furthermore, analysis shows that the pooling based model is the most efficient. Thus, we achieved state-of-the-art results in the code generation task. Specifically, on Hearth Stone dataset we outperformed all the previous methods by 78.9 BLEU. In addition, we provided some advice to researchers that they must consider practical reality of target tasks, and do not fall into the trap of using complicated deep learning models blindly. In conclusion, our work has theoretical significance and practical value in the field of software engineering.

In the future, we plan to adopt more models in code generation tasks, aiming at seeking ways to improve the performance. However, there is a syntactic difference between various program languages. Thus, we also plan to apply CDS models into more code generation datasets.

REFERENCES

- [1] W. Ling, P. Blunsom, E. Grefenstette, K. M. Hermann, T. Kočiský, F. Wang, and A. Senior, "Latent predictor networks for code generation," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2016, pp. 599–609.
- [2] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2017, pp. 440–450.
- [3] M. Rabinovich, M. Stern, and D. Klein, "Abstract syntax networks for code generation and semantic parsing," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2017, pp. 1139–1149.
- [4] L. Dong and M. Lapata, "Language to logical form with neural attention," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2016, pp. 33–43.
- [5] N. Kushman and R. Barzilay, "Using semantic unification to generate regular expressions from natural language," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2013, pp. 826–836.
- [6] M. Raza, S. Gulwani, and N. Milic-Frayling, "Compositional program synthesis from natural language and examples," in *IJCAI*, Q. Yang and M. Wooldridge, Eds. AAAI Press, pp. 792–800.
- [7] T. Lei, F. Long, R. Barzilay, and M. Rinard, "From natural language specifications to program input parsers," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2013, pp. 1294–1303.
- [8] M. M.-H. Tso, "Context-sensitive template engine," Jul. 4 2000, uS Patent 6,085,201.
- [9] T. Veldhuizen, "Method and apparatus for generating inline code using template metaprograms," Nov. 10 1998, uS Patent 5,835,771.
- [10] S. A. Hayati, R. Olivier, P. Avvaru, P. Yin, A. Tomasic, and G. Neubig, "Retrieval-based neural code generation," *CoRR*, vol. abs/1808.10025, 2018.
- [11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014.
- [12] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2015, pp. 1412–1421.
- [13] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1243–1252.
- [14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008.
- [15] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.
- [16] N. Kitaev and D. Klein, "Constituency parsing with a self-attentive encoder," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2018, pp. 2676–2686.
- [17] Z. Sun, Q. Zhu, L. Mou, Y. Xiong, G. Li, and L. Zhang, "A grammar-based structural CNN decoder for code generation," *CoRR*, vol. abs/1811.06837, 2018.
- [18] M. Iyyer, V. Manjunatha, J. Boyd-Graber, and H. Daumé III, "Deep unordered composition rivals syntactic methods for text classification," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, 2015, pp. 1681–1691.
- [19] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, 2017, pp. 427–431.
- [20] D. Shen, G. Wang, W. Wang, M. Renqiang Min, Q. Su, Y. Zhang, C. Li, R. Henao, and L. Carin, "Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2018, pp. 440–450.
- [21] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014, pp. 1746–1751.
- [22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.
- [23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.
- [24] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002.

Safe-by-Design Development Method for Artificial Intelligent Based Systems

Gabriel Pedroza, Morayo Adedjouma

CEA LIST, Department of System and Software Engineering
P.C. 174, Gif-sur-Yvette, 91191, France
{firstname.lastname}@cea.fr

Abstract

Albeit Artificial Intelligent (AI) based systems are nowadays deployed in a variety of safety critical domains, current engineering methods and standards are barely applicable for their development and assurance. The lack of common criteria to assess safety levels as well as the dependency of certain development phases w.r.t. the chosen technology (e.g., machine learning modules) are among the identified drawbacks. In addition, the development of such engineering methods has been hampered by the emerging challenges in AI-based systems design mainly regarding autonomy, correctness and prevention of catastrophic risks. In this paper we propose an approach to conduct a safe-by-design development process for AI based systems. The approach relies upon a method which benefits from a reference AI architecture and safety principles. This contribution helps to address safety concerns and to comprehend current AI architectures diversity and particularities.

Index Terms—safe-by-design, AI, safety, engineering

I. Introduction

The implementation of Artificial Intelligence (AI) based systems has progressed in many aspects. The technology shows increasing levels in tasks automation and adaptation to the context. For instance, for self-driving vehicles, we can cite the DriveMe project on Volvo XC90 series launched in 2013 [1], the Autopilot technology integrated in Tesla vehicles since 2013 [2], and the Waymo project of Google which conducted the world's first self-driving ride on public roads [3] claiming maximum autonomy level (5). Several instances of systems based upon AI technology can be found in the literature, e.g., [4], [5]. The main concerns addressed by those architectures are related to (1) the limits imposed by sensors' detection, (2) the heuristic nature of

machine learning (ML) and deep learning (DL) techniques, and (3) the variability and complexity of context scenarios. Whereas current technology and implementations show approach feasibility and provide some solutions to referred concerns, they are mostly committed to improve system's autonomy letting aside other aspects of the engineering process. In general for AI-based systems development, a variety of engineering techniques are applied and combined with almost empirical parameters, choices which are finally tuned to optimize performance [6]. Despite the engineering process has proven certain effectiveness, it also shows some drawbacks regarding safety assessment. In particular, the lack of a comprehensive process to settle common criteria and thresholds for safety evaluation and certification. To our knowledge, the problem is quite hard to solve and no current approach overcomes related issues: an AI-based system should integrate sensor devices with limited - sometimes opaque - detection capabilities ($\leq 90\%$ in average), deploy algorithms to identify and properly react to complex, rather unforeseen, situational scenarios and, on top of that, ensure negligible likelihood of occurrence for critical hazards and disfunctioning. Moreover, whereas for typical development methods, like the V-cycle, the phases and their sequencing are almost static, it is observed that for AI-based systems, the nature of certain engineering phases and their order may vary. Some of the factors for that to occur are the dependency of the engineering process on the AI technology choices, on the knowledge bases - used for learning - and on their maturity (representativeness of data sets, events, phenomena). Although, it is consensual that safety analyses should be conducted as early as possible and all along the life-cycle, the few initiatives on that respect, like ISO/IEC 23053 [7], are still work in progress. Others, like ISO 21448 [8], provide insights on a safety-integrated process for autonomous vehicles without settling generic criteria suitable for other application domains. Consequently, current

standards landscape is barely applicable to the growing space of AI-based systems, and new safety methods and analysis processes are required to develop them. To tackle the referred issues, the main contribution of this paper is an overall generic iterative (OGI) method to conduct safe-by-design development of AI-based systems relying upon a generic architecture and safety principles.

The rest of the paper is structured as follows. In Section II, the reference AI architecture is introduced. In Section III, some safety relevant aspects to AI-based systems are highlighted. The OGI process for AI-based systems development is described in Section IV, including safety assessment phases. In Section V, the safe-by-design method is applied to the development of an autonomous system. Some related works are explained in Section VI. Finally, a discussion and work perspectives come in Section VII.

II. Generic Reference AI-based Architecture

The specification of a process development for AI-based systems should consider the following particularities:

- *Engineering process dependent on AI technology:* Subsystems or components implementing ML/DL modules are based upon parameters which may require to be set during conception, design, implementation and validation phases. For those subsystems and components, a learning phase should be introduced whereas for other typical (non-ML-based) subsystems and components, the learning phase is non-existing.
- *Engineering process dependent on knowledge bases:* The learning phases strongly depend upon target objectives and external knowledge bases (KBs). For many complex AI-based systems, an iteration on design parameters may be necessary after a validation campaign, *e.g.*, to adjust detection ranges and accuracy. Detailed requirements cannot be elicited before knowing the effectiveness of knowledge bases, ML/DL techniques, and parameters choices.
- *Knowledge bases maturity:* Due to previous aspects, the AI-based systems development shall not only depend upon ML/DL techniques and development methods, but also on building up knowledge bases (KBs) and making them evolve so as to improve coverage and accuracy of objects, events, and phenomena detection. Referred KBs shall be useful for detection and for high level reasoning, *e.g.*, reasoning based upon intuition [9].

We propose to integrate the previous specificities from early stages of design. Since a huge diversity of architectures and process cycles currently exist, the specification of a development process should be as generic and comprehensive as possible. We find suitable to first introduce a generic AI-based architecture aiming to cover

most of them. The Figure 1 illustrates the coarse domains composing our reference architecture which are briefly described in the following items and in section IV.

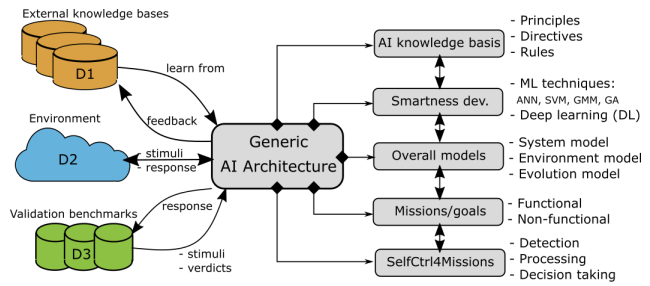


Fig. 1. Generic reference AI architecture

- *External knowledge sources:* this domain comprises KBs required for training ML/DL modules, for their implementation and validation. At the beginning, these external KBs are not part of the system, but they are integrated as a logical component during the engineering process, for instance, after generation of training sets (*e.g.*, via the feature vectors).
- *Missions/goals:* this domain of the architecture covers the fulfillment of functional and non-functional goals and missions of the system. Certain missions and goals may not require the deployment of ML/DL modules. Thus, they can be deployed by components and subsystems relying upon typical technology.
- *AI knowledge basis:* this domain includes components for the fulfillment of principles, directives and rules which guide the AI-behavior. The modules pertaining to this domain provide a basis upon which missions and goals can be accomplished by the system.
- *Smartness development:* the smartness of the system relies, at least, upon two layers of reasoning. The first layer is in charge of detection of external objects, phenomena, and scenarios. The second layer includes monitoring of system status, self-positioning and reacting to external conditions according to missions and AI-principles. The development of system smartness depends on AI techniques like ML and DL.
- *Overall models:* the development of autonomy and smartness demands an understanding of environmental and internal system elements. As for external elements, a model of the environment is to be settled. Among others, this model allows the interpretation of external stimuli. As for internal elements, the system should be able to have a comprehensive model of itself. Among others, this model allows to assess system self-status. Both models capture the current capabilities of the overall system. To ensure certain independence, the AI-system should be able to learn

from new stimuli and situations and it should be able to integrate and deploy new capabilities.

- *Self-control*: this domain includes the functions deployed to realize autonomy during missions and goals accomplishment. A typical functional path comprises sensors→controllers→actuators that respectively support detection, processing and decision-taking.

III. Safety Concerns for AI-based Systems

Safety is one of utmost relevant concerns when designing AI-based systems. However, it is also a vast and complex subject considering the multiple applications domains where they can be deployed and their related specificities. The main safety related activity affected by the specificities of AI-based systems is the hazard analysis at the concept phase of the system. Indeed, the implementation of ML/DL components rise new concerns regarding emerging categories of hazardous events. Referred hazardous events exhibit an increased risk level (which may even be catastrophic) due to the machine overtaking over former human-based activities. Along with more autonomy, the transfer of duties to AI-based algorithms implies no further human interaction as safety barrier in case of hazards. In addition, certain typical safety criteria like redundancy do not suffice anymore to ensure expected levels of availability and also accuracy. The specific aspects addressed in this paper are described in line. Some of these aspects have already been highlighted in emerging safety standards like ISO 21448 [8] (see Figure 2).

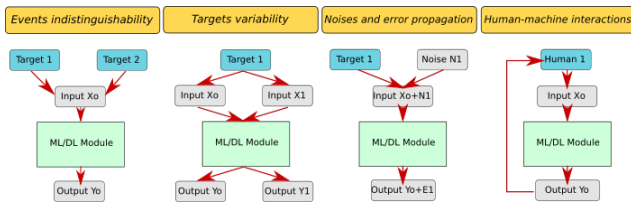


Fig. 2. Hazard events related to ML modules

- *Events indistinguishability*. This issue is mainly due to physical limits of sensors. In particular, because real and virtual images are practically indistinguishable. For instance, the detection of a stop signal can be easily faked by a photo of the same. Light rays and other natural electromagnetic signals used for detection can be reproduced by different objects.
- *Targets variability*. Many objects, elements and phenomena to be detected by AI-based systems exhibit certain variability. Although AI-based systems should cope with referred variability, questions arise if the system may face unforeseen situations beyond its

limits. For instance, sudden deterioration or disfunctioning of context signs may lead to wrong detection of objects.

- *Noises and error propagation*. Along with variability issues, the noises added by the background and environmental phenomena increase hazards impact, in particular in case of dismissed/miscalculated noises and errors propagated through the system. Environmental conditions (e.g., solar winds, snow) may impose additional constraints to system operation and behave as background noises.
- *Human-machine interactions harmonization*. The operation of AI-based systems in real environment is quite novel. Whereas risks related to machine-to-machine or machine-to-environment interactions can be assessed during design phases, predict the outcomes of human-to-machine interactions is far more complex [10]. Although an harmonization phase can be conducted, complex human reactions (e.g., psychological, societal, political) are difficult to assess.

IV. Safe-by-Design Method for AI Systems

In this section, we introduce a method to develop the architectural domains specified in section II. The safety aspects described in section III are integrated into the cycle. The method is illustrated in Figure 3 and is described in the following subsections. The method is iterative and several phases can be conducted in parallel even if interdependencies may appear.

A. AI-based systems development method

The main phases of the OGI process are briefly described in the following items.

- 1) *Missions and goals*. This phase comprises the specification of missions and goals the system should accomplish. The specification mostly targets functional requirements of a typical engineering phase. However, as for safety-critical systems, it also includes the management of non-functional requirements.
- 2) *AI principles structuring*. This phase covers the specification and structuring of AI principles upon which the system relies. As for typical requirements stages, the formalization and validation of consistency between AI principles is a major stake. This principles are processed afterwards in phase 4.
- 3) *De-compositional analysis*. This phase is dedicated to decompose, refine and structure missions and goals up to obtain a layer including detailed functions. This phase mostly follows a typical process of design refinement. However, notice that a subset

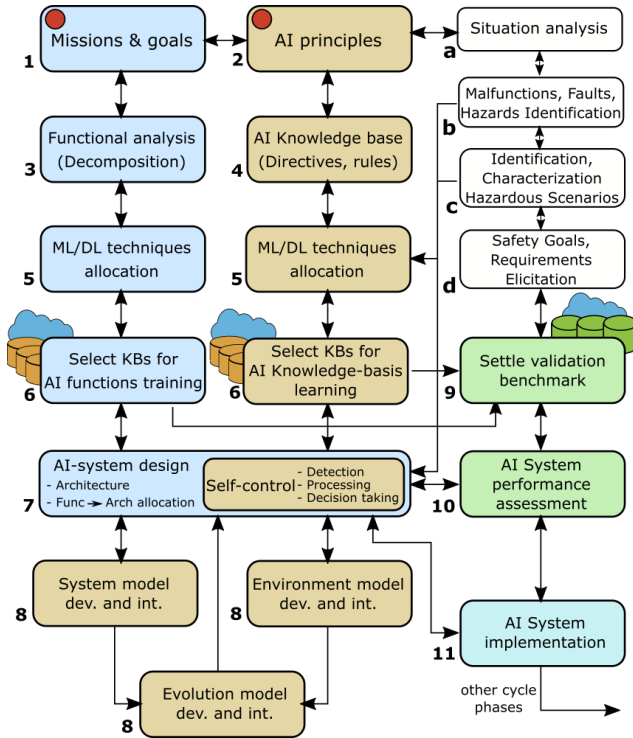


Fig. 3. Method for AI systems development

of functions and blocks are to be carried out by ML/DL-based modules as described in phase 7.

- 4) *AI knowledge basis structuring.* The structured AI principles in phase 2 are first decomposed into a set of high level directives which are afterwards refined up to obtain a set of rules including specific system behaviors. The elicited rules play the role of policies which help to guide - or even enforce - system behavior when needed.
- 5) *Allocation of ML/DL techniques.* This phase is dedicated to allocate concrete ML/DL techniques and modules to the functions and blocks elicited in phase 3, “*De-compositional analysis*”, and to the behaviors elicited in phase 4, “*AI knowledge basis structuring*”. The allocation should (1) accomplish the missions and goals of the system and (2) ensure the compliance with rules refined from AI principles and directives.
- 6) *Knowledge bases selection.* Once the allocation of ML/DL techniques is finished, the KBs for training the respective modules are selected. Of course, building up new or dedicated KBs may be necessary. The KBs are the basis to generate outside-world and system models as well as the evolution model by performing the training tasks (see phase 8).
- 7) *Detailed AI architecture design.* The detailed design

of the AI-based system comprises at least three tasks or sub-phases. The first one consists in proposing the architecture to support the functions specified in previous phases 4 and 5, including allocations. In the second sub-phase, a distribution of functions over the support architecture is conducted. This task covers the exploration of the design space. In the third and last sub-phase, the first layer of intelligence is developed by training the ML/DL modules, *i.e.*, the self-control functions for detection, processing and decision taking.

- 8) *Overall models development and integration.* The second layer of intelligence is developed in this phase: the AI-system should be able to learn from new stimuli and situations. The models of the external world and the system itself have been partially developed and integrated during previous training tasks. Upon outside-world and system models, an evolution model should be elaborated and integrated relying upon (1) specific principles governing AI autonomy, (2) techniques and metrics to assess and filter new stimuli and situations (*e.g.* adaptive decision making [11]) and (3) techniques to integrate new filtered knowledge so as to grow up system and environment models.
- 9) *Settle validation benchmark.* After the allocation of ML/DL techniques, a benchmark for validation can be settled. The benchmark includes a set of target objectives used to assess performance and emit verdicts. The target objectives are refined from missions, goals, AI principles and other requirements *e.g.*, safety requirements. The benchmark also includes the data sets selected or generated to test the performance of ML/DL modules. When applicable, the validation benchmark can be based upon typical validation and verification techniques like testing, simulation, and formal verification.
- 10) *AI system performance assessment.* The system performance is evaluated relying upon the validation benchmark. An iterative process starts in order to fulfill the target objectives non satisfied after the first validation campaign.
- 11) *AI system implementation.* Along with typical technology components, the components including trained ML/DL modules and their parameters are deployed in this phase.

B. Integration of safety aspects into the OGI cycle

The assessment of the specific safety aspects of ML/DL components, introduced in Section III, demands dedicated principles. Referred safety aspects are integrated into the OGI cycle (see Figure 3) as follows:

- a) **Situations analysis.** This activity corresponds to a typical operational situation analysis as in hazard analysis [12]. However, special attention is given to situations in which autonomous functions operate and are at stake.
- b) **Malfunctions, faults and hazards related to ML/DL modules.** Along with classical malfunctions, faults and hazards, the aspects referred in subsection IV-A are considered as sources of new potential malfunctioning and hazardous behaviors. For their identification, each safety aspect is related to the architecture parts potentially impacted. For instance, regarding *events indistinguishability*, ML/DL functions, blocks and components carrying out detection are to be considered; for object *variability* and *noises and error propagation*, the reasoning layer used for environment interpretation is in cause, and for the *human-machine interaction harmonization*, the decision-taking reasoning layer is targeted. In addition, the heuristic nature of ML/DL algorithms as well as their dependency to external KBs demand the definition of a malfunctioning matrix including false positives and false negatives occurrence. The autonomy of an AI-based system can be assessed based on its ability to cope with multiple components malfunctioning and hazards (accidental, misuse, natural). Concretely, for each component C_i and target T_j with background B_j the probability of error should be minimized:

$$P[ErrorC_{(i,j)}] := P[FalsePos_{(i,j)}] + P[FalseNeg_{(i,j)}],$$

$$FalsePos_{(i,j)} := \cup_j \{C_i(Accept, B_j)\},$$

$$FalseNeg_{(i,j)} := \cup_j \{C_i(Reject, T_j)\}.$$

Other factors of components failure can be considered, in particular the failure rate λ_i along with the probability of failure over time $P[FailC_{(i,t)}] = \lambda_i e^{-\lambda_i t}$. Thus, the overall probability of component disfunctioning can be calculated by:

$$P[DisfC_{(i,j,t)}] = \omega_1 P[FailC_{(i,t)}] + \omega_2 P[ErrorC_{(i,j)}],$$

$$\omega_1 + \omega_2 = 1.$$

- c) **Identification and characterization of hazardous scenarios.** Along with classification of typical hazardous scenarios, the hazardous scenarios involving ML/DL components should be characterized. Notice that, the typical scenarios referred in ISO 26262 [13] as “reasonably foreseeable misuse” are no longer under human control and consequently should also be reclassified. To do so:

- Scenarios involving ML/DL components are defined as combinations of external stimuli, system response (internal stimuli) and malfunctions.
- For each scenario S_k , subsets of KBs are selected to evaluate the performance of involved ML/DL components ($\{C_i\}$):
 - Data/features characterizing legitimate targets T_j (true positives)

- Data/features characterizing targets’ backgrounds B_j (true negatives)
 - The likelihood of each scenario $P[S_k]$ is estimated by combining the disfunctioning probabilities $\{P[DisfC_{(i,j,t)}]\}$ of involved components $\{C_i\}$ according to the architecture structure and relying upon basic probability theory.
- d) **Safety goals elicitation.** Elicitation of safety goals based upon ASIL levels is rather coarse and thus inadequate considering the current nature of hazardous scenarios $\{S_k\}$. Since a huge diversity and complexity of hazardous scenarios are possible, three cases are identified from which detailed requirements can be elicited:
- The hazardous scenario S_k can be associated to a concrete behavior which is formalized and monitored. A monitoring formula ϕ including safety threshold θ can thus be elicited. For instance, in the case of autonomous vehicles, a formula for minimal safety distance between them is settled: $\phi \geq \theta$. In this case, the detailed requirements are elicited in terms of a maximum threshold violation: $P[\phi < \theta] \leq \delta$.
 - The hazardous scenario S_k can be associated to a concrete behavior but deriving a safety monitoring formula is not evident. If the behavior can be formalized and a validation test-bench can be settled, then performance tests are conducted. In this case, the detailed requirements can also be elicited in terms of maximum probability of error given by $\{P[DisfC_{(i,j,t)}]\}$.
 - The hazardous scenario or the associated behavior are complex. Conducting performance tests is unfeasible. In this case, the behavior can be formalized and a validation test-bench can be settled relying upon simulation. For sophisticated simulation test-benches, the requirements can be elicited as in previous case, *i.e.*, via $\{P[DisfC_{(i,j,t)}]\}$. Otherwise, safety requirements may depend upon simulation scenarios.

V. OGI Method Tool Support and Evaluation

The modeling and safety assessment framework of the OGI method is implemented in Sophia, a model-based toolset integrated with Eclipse Papyrus editor for UML/SysML models [14]. Sophia uses Papyrus extension mechanisms to support safety and reliability analyses like HARA, FMEA, FTA. We evaluate the OGI method on an ongoing industry-academy experimentation of autonomous shuttles deployment on a sensitive site, as described in [12]. Due to lack of space, the evaluation only focus

on selected method phases. A critical mission of the autonomous shuttle is to adopt a safe reaction in presence of an obstacle on its trajectory. The selected scenario S_k corresponding to the mission is presented in the sequence diagram in Figure 4. The scenario S_k involves the *Percep-*

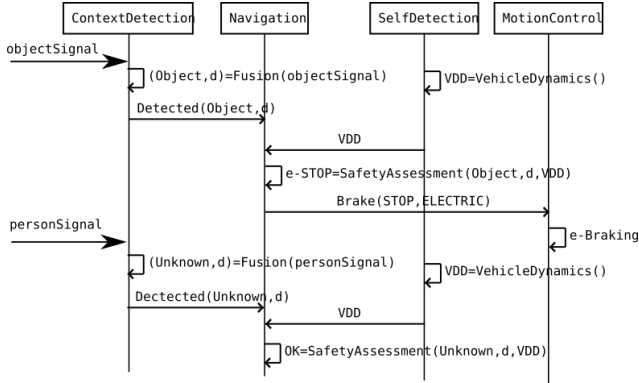


Fig. 4. Shuttle safety critical scenario

tion (i.e. *ContextDetection* and *SelfDetection*), *Navigation* and *Motion control* functions of the system. In the first exchanges, the shuttle detects an object on its route, and is able to brake at safe distance to avoid an accident. In the second iteration, although the system detects an obstacle on its path, it is not recognized as a person due to malfunctioning. Consequently, the shuttle does not take appropriate decision to avoid the collision. Figure 5 presents an excerpt of the architecture design showing the allocation of the scenario functions. The *SelfDetection*

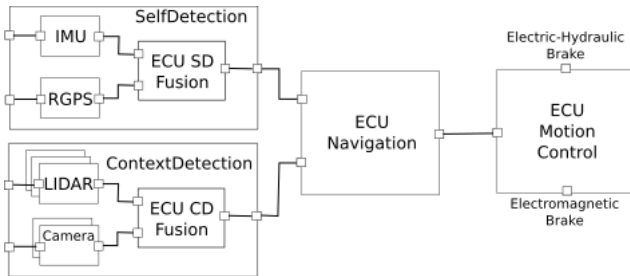


Fig. 5. Excerpt of shuttle architecture design

function (see *Detected()* in Figure 4) is realized by an inertial measurement unit (IMU), a relative GPS (RGPS) and a ML/DL component for data fusion (see *Fusion()* in Figure 4), the latter computes the vehicle dynamics e.g. speed, acceleration, momentum (see *VehicleDynamics()* in Figure 4), thus settling the model of the system itself. The *ContextDetection* relies upon a set of LIDARS and Cameras, and upon an ML/DL data fusion component which structures a model including speed and position of objects within the surrounding vehicle's environment.

The *Navigation* function is also an ML/DL component that interprets the scene and take a decision according to safety and AI-based directives (see *SafetyAssessment()* in Figure 4). The decision is afterwards sent to an ordinary *Motion control* component for action-taking. Decision Trees, Support Vector Machines, Gaussian Mixture Models, Fuzzy Logic, and other unsupervised machine learning are example of techniques to be used for the ML/DL components. In order to acquire their capability of detection, scene interpretation and decision taking, the ML/DL components have been trained with dedicated KBs prior to architecture's definition. The shuttle process development follows the phase 1 to phase 7 shown in Figure 3. Notice that, in the hazardous scenario in Figure 4 (failure in person's detection), since the system was neither able to perform a collision avoidance maneuver, it presupposes that an intelligence layer is missing: the evolution model to face unknown situations is still to be constructed (see Figure 3, Phase 8). The evaluation of the hazardous scenario is conducted based upon our proposed safety assessment method. To do so, an existing hazard analysis of the shuttle [12] is considered which covers the phases a, b, c, d of the OGI cycle (see Figure 3). The concerning fault in the scenario is the non recognition of a person as a such. The overall probability of the hazardous scenario $P[S_k]$ (see section IV) can be calculated in terms of the probability of disfunctioning for each component $P[DisfC_{(i,j,0)}]$ as follows:

$$P[S_k] := \lambda_{LIDAR}\lambda_{Camera} + P[DisfECUCD] + \lambda_{IMU}\lambda_{RGPS} + P[DisfECUSD] + P[DisfECUNavigation] + \lambda_{MotionControl}.$$

A detailed requirement for the scenario can now be specified as $P[S_k] \leq \theta$ which allows to settle a target for the validation benchmark and assess the performance of the system (covering phase 9 and phase 10 in Figure 3).

VI. Related Work

Several works have been presented to identify and integrate safety during AI systems development by industry and academy. In [15], a survey of main AI safety problematics is provided, among them, wrong or cost-ineffective objective functions, and ineffective learning phases. In [16], a preliminary study is conducted discussing different aspects of AI systems' safety. The authors propose the application of classical formal verification principles to AI systems design, like randomized formal methods for training, in combination with design space exploration guided by safety criteria. The authors in [17] present a design strategy for autonomous architectures which maps safety requirements over a global control module. A lot

of proposals exist addressing the optimization of ML/DL based modules performance, *e.g.*, [18]. All previous cited approaches either only cover specific phases of the engineering cycle or are quite specific to a given application domain. On the contrary, the work in [11] presents a holistic approach for autonomous systems development addressing similar safety concerns as in this paper, and also relying upon a layered architecture. Unfortunately, it does not propose any criterion for evaluation of hazardous scenarios nor for safety goals elicitation. The theoretical perspective for a safe AI cycle presented in [19] is quite aligned to the one in this paper, with regard to the integration of KBs into the engineering loop targeting data and software diversity. Being a theoretical work, the elicitation of precise system requirements is nonetheless not covered.

VII. Discussions and Perspectives

This paper presents an overall iterative generic (OGI) method for development of AI-based systems. The method is based upon reference architecture domains dependent upon KBs, environment model, validation benchmarks, among others. In addition, the method integrates assessment activities to tackle specific safety-critical aspects of such systems. The assessment provides an enhancement of the typical hazard analysis method to infer safety goals. In particular, it yields the disfunctioning likelihood of an AI-based component considering the typical failure rate added up with the error probability of ML/DL modules. By applying the OGI method to the autonomous shuttle, a multi-factor uncertainty was identified intervening at different levels of AI systems design. A first uncertainty comes from the accuracy and maturity of external KBs which impact the learning process and performance of ML/DL components. The use of fine-grained approaches, like data diversification [19], is promising to overcome this issue. A second uncertainty is the difficulty to apprehend the infinite usage-scenarios space resulting from a “continuum” of possible environmental-operational contexts, variants and configurations. This often leads to poor system-context specifications not representative enough to characterize the scenarios space. A current trend for this shortcoming is to settle enough specificities, from a safety point of view, to define categories of emblematic scenarios [20] useful in particular for benchmarking but without guarantee of coverage exhaustiveness. The third uncertainty factor comes from the performance limits of AI-based components, *e.g.*, sensors blinding or other still unveiled environment events. Interpretation and decision-taking layers are also at stake when arbitration algorithms face contradictory directives and must solve them in critical scenarios, *e.g.*, between safety requirements and AI-knowledge basis (principles, directives, rules). As of today, there is no ethical solution

that has reached consensus on this sensitive issue. Finally, since AI-based systems may still be unable to properly react to changing environments (as in the case study in section V), deploy new capabilities in real time is of utmost importance to ensure true systems intelligence and autonomy. In future work, we plan to conduct larger-scale application of the OGI method on others safety-critical AI-based systems to strengthen our conclusions and enforce approach validity. We are also assessing the applicability of other standard-preconceived methods, like FMEA and FTA, in the context of AI-based systems. We would further like to complete the OGI method to cover latter stages of the development cycle, *i.e.*, testing and validation.

References

- [1] Volvo. (2018) Drive me project. [Online]. Available: <https://www.volvocars.com/intl/about/our-innovation-brands/intellisafe/autonomous-driving/drive-me>
- [2] Tesla. (2018) Tesla autopilot. [Online]. Available: <https://www.tesla.com/autopilot>
- [3] Google. (2018) Waymo, google project. [Online]. Available: <https://waymo.com>
- [4] S. Dersten *et al.*, “An analysis of a layered system architecture for autonomous construction vehicles,” in *(SysCon) 2015 Proceedings*, April 2015, pp. 582–588.
- [5] A. Yavnai, “Distributed decentralized architecture for autonomous cooperative operation of multiple agent system,” in *AUV’94 Proceedings*, July 1994, pp. 61–67.
- [6] G. Pedroza *et al.*, “A speaker verification system using svm over a spanish corpus,” in *2009 Mexican International Conference on Computer Science*, Sep. 2009, pp. 381–386.
- [7] *Framework for Artificial Intelligence (AI) Systems Using Machine Learning (ML)*. ISO, 2019.
- [8] (2019) Road vehicles - safety of the intended functionality.
- [9] A. Rauber *et al.* (2019) Transparency in algorithmic decision making. [Online]. Available: <https://ercim-news.ercim.eu/images/stories/EN116/EN116-web.pdf>
- [10] F. M. Favaro, N. Nader, S. O. Eurich, M. Tripp, and N. Varadaraju, “Examining accident reports involving autonomous vehicles in California,” *PLOS ONE*, vol. 12, pp. 1–20, 2017.
- [11] A. Aniculaesei *et al.*, “Towards a holistic software systems engineering approach for dependable autonomous systems,” in *SEFAIS ’18 Proceedings*, 2018, pp. 23–30.
- [12] M. Adedjouma *et al.*, “Representative safety assessment of autonomous vehicle for public transportation,” in *ISORC 2018 Proceedings*, 2018, pp. 124–129.
- [13] *ISO 26262: Road Vehicles - Functional Safety*. ISO, 2011.
- [14] M. Adedjouma and N. Yakymets, “A framework for model-based dependability analysis of cyber-physical systems,” in *19th IEEE International Symposium on High Assurance Systems Engineering (HASE) 2019*, Hangzhou, China, January 3–5, 2019, 2019.
- [15] D. Amodi *et al.*, “Concrete problems in ai safety,” *CoRR*, 06 2016.
- [16] S. A. Seshia and D. Sadigh, “Towards verified artificial intelligence,” *CoRR*, vol. abs/1606.08514, 2016.
- [17] C. Molina *et al.*, “Assuring fully autonomous vehicles safety by design: The autonomous vehicle control AVC module strategy,” *(DSN-W) 2017 Proceedings*, pp. 16–21, 2017.
- [18] C. Huang *et al.*, “A GA-based feature selection and parameters optimization for support vector machines,” *Expert Systems with Applications*, vol. 31, no. 2, pp. 231 – 240, 2006.
- [19] R. Ashmore *et al.*, “Rethinking diversity in the context of autonomous systems,” in *SSS 2019 Proceedings*, 2019, pp. 175–192.
- [20] A. Jang *et al.*, “A study on situation analysis for asil determination,” vol. 3, 01 2014.

Augmenting App Reviews with App Changelogs: An Approach for App Reviews Classification

Chong Wang*, Tao Wang*, Peng Liang
School of Computer Science
Wuhan University, China
{cwang, liangp}@whu.edu.cn

Maya Daneva, Marten van Sinderen
School of Computer Science
University of Twente, the Netherlands
{m.daneva, m.j.vansinderen}@utwente.nl

Abstract—Recent research on the automatic classification of app reviews either focused on grouping app reviews into categories relevant to software evolution, or employed app reviews as the only research data to improve app reviews classification. Although it was reported that app review classification can benefit from supplementing user reviews with the data from other sources, only a few studies employed app changelogs for this purpose. This paper explores how to augment app reviews with changelogs to improve the accuracy and performance of classifying functional and non-functional requirements in app reviews. Specifically, we propose AUG-AC as an approach to extract feature words from app changelogs and construct the augments for app reviews. Next, we designed a series of experiments to evaluate our approach, varying in the length of AC-based augments for app reviews. The results show that AUG-AC outperforms the existing method by using app changelogs as a source of data next to app reviews.

Keywords—app reviews, app changelogs, requirements analysis, machine learning, data-driven requirements engineering

I. INTRODUCTION

With the rapid growth of mobile applications, massive sets of data are provided by the crowds. Particularly, app reviews, a type of explicit feedback from the users, have been recognized as an important source of user requirements for app updating and maintenance [1-3]. However, current research [1-2] mainly concentrated on how to extract features or topics from a large number of app reviews and then classify these topics into categories relevant to software evolution. Several studies [4-7, 14] also explored the use of user feedback from other sources in requirements elicitation. For example, Vu et al. [4] employed user reviews of packaged software in Amazon to pre-extract phrases for mining user opinions from app reviews, while Jiang et al. [5] combined product reviews from Amazon with app reviews as the research data. These authors [4-5] observed that user reviews of software have similar characteristics to app reviews: (1) the number of reviews is increasing rapidly every day; (2) review texts contain many noise words, including emoji, non-English words, misspelled words, user-defined abbreviations; and (3) most reviews are non-informative (as reported in [6], only around 30% of app reviews are informative for app updates). To reduce the manual effort in filtering out non-informative samples and identify valuable information for developers, this paper explores if other information of apps, especially the pieces with less noise (i.e. app changelogs), could be a significant help.

App changelogs are posted by software vendors regularly in weeks or months. These official texts are written in a standardized way and comprise primary changes of the releases. A 2018 ICSE study [7] has successfully employed app changelogs to identify emerging issues in app reviews. We were motivated by these findings, and set out to explore how to use official app changelogs to improve the accuracy and performance of classifying requirements in app reviews. Especially, this paper intends to explore how to make use of app changelogs in the automatic classification of app reviews from the perspective of requirements types, and finally aid developers in the maintenance and updating of apps.

The paper is structured as follows. Sect. II is on related works. Sect. III presents our approach. Sect. IV reports on the experimental results evaluating and comparing the accuracy and performance of our approach with others. Sect. V discusses our findings. Sect. VI is on validity threats. Sect. VII concludes.

II. RELATED WORK

Considering the automatic classification of app reviews, some researchers proposed categories relevant to software maintenance and evolution. Maalej et al. [1] introduced several probabilistic techniques to classify app reviews into four categories, i.e. bug reports, feature requests, user experience, and text rating. Guzman et al. [2] proposed seven categories relevant to software evolution, viz. bug report, feature strength, feature shortcoming, user request, praise, complaint, and usage scenario. The categories proposed in these two studies partially overlap, since the authors intended to help app vendors and developers filter critical reviews relevant to different aspects of software maintenance. Other researchers considered the categories of app reviews from the perspective of requirements types. In particular, our previous works in [8-9] employed classic machine learning algorithms to identify and classify functional and non-functional requirements (FRs and NFRs) from app reviews. Another similar study [10] performed automatic analysis on app reviews for NFRs elicitation and prioritization. In all these studies, however, app reviews were the only type of research data to apply and compare specified classifiers.

To the best of our knowledge, only very few studies employed research data from other sources to analyze app reviews. Those other sources include user reviews of software [4], of products [5], and app descriptions [14], etc. However, both [4, 5] aimed to extract and cluster user opinions, rather than

*: The authors contributed equally to this work.

classifying requirements into different categories. In [14], Liu et al. used app descriptions, another typical data in app stores, to guide the analysis of user reviews. Gao et al. [7] used app changelogs to identify emerging issues from app reviews, instead of identifying and classifying requirements.

In contrast to these previous studies, the focus of our work is mainly on how to augment app reviews with app changelogs in order to improve the accuracy and performance of classifying FRs and NFRs from app reviews.

III. OUR APPROACH

To employ official app changelogs for the automatic identification and classification of requirements from app reviews, we propose an approach, called AUG-AC, to AUGment app reviews with the text feature words extracted from App Changelogs (AC). In this section, we give an overview of AUG-AC to explore how to improve the automatic classification of app reviews by employing official app changelogs. Each step of our approach will be detailed in a subsection.

The experimental data collected and manually labeled in our previous work [13] will be reused to evaluate the performance and AUC-AC. The dataset includes 6000 app review sentences of three apps (one from Apple App Store and two from Google Play) and 2024 app changes filtered from 2005 official changelogs of 30 apps (3 *categories* \times 10 *apps* in Apple App Store). As described in [13], these app review sentences and changes were labeled with six types of requirements, including four types of NFRs defined in ISO 25010 [11] (i.e. *Usability*, *Reliability*, *Portability* and *Performance*), FR, and ‘Others’ - the type referring to those review sentences and app changes that fit neither FRs nor the four NFRs listed above.

A. Overview

Our approach consists of four main steps, as Figure 1 shows.

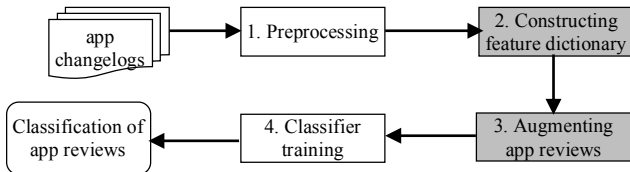


Figure 1. Overview of AUG-AC approach, focusing on the steps in grey.

The first step is to preprocess the sampled app reviews and changelogs to get respective text feature words of each requirements type (Sect. III.B). The second step (which is the one foci of our work) creates a AC-based feature words dictionary for augmenting app reviews (Sect. III.C). The third step (the other focus of our work) augments app reviews with the text features extracted in the second step to train the classifier and group the app reviews into six pre-specified types of requirements (Sect. III.D). In the last step, those augmented app reviews generated in Sect. III.D construct the training set of the specified classifier. The accuracy of applying this classifier to categorizing app reviews in the test set will be evaluated by the standard metrics Precision, Recall and F-measure (Sect. III.E).

B. Text Preprocessing

In this step, multiple Natural Language Processing (NLP) techniques were applied to the text of app review sentences and changes. Specifically, Natural Language Toolkit (NLTK) was adopted to perform stopword removal, punctuation removal, and lemmatization.

Next, considering each type of requirements, we intended to extract text feature words from the app changelogs and treat them as the candidate augmented words for the app reviews labeled as this type. In general, the concerns of app changelogs grouped in each type of requirements may not always be well represented by the frequency and importance of a word. Thus, for each type of requirements, Latent Dirichlet Allocation (LDA) was employed to extract text feature words in app changelogs. By applying LDA, app changelogs labeled as a certain type of requirements into can be clustered into one topic and produce topic words for this cluster (i.e. each type of requirements). In this work, topic words of each cluster form the initial set of text feature words to be augmented to those app reviews that are labeled as the corresponding type of requirements.

C. Constructing AC-based Feature Dictionary

This step aims to construct an AC-based feature dictionary, consisting of the text feature words that was extracted from app changelogs and to be augmented to app reviews. As already said in the beginning of Sect. III, six types of requirements have been specified as the category labels for both app reviews and changelogs. Accordingly, for each requirements type i , a AC-based feature dictionary D_i is needed (1) to store the text feature words extracted from app changelogs typed as i , and (2) to provide candidate AC-based augmented words for app reviews typed as i . In this paper, D_i is initialized as a set containing the top 20 features words t_{ij} extracted from app changelogs labeled with requirements type i .

Algorithm 1: Constructing AC-based Feature Dictionary	
Input:	D_i – initial AC-based feature dictionary.
Output:	D_i' – extended AC-based feature dictionary.
1	for each requirements type i
2	import D_i ;
3	Insert D_i to D_i' ;
4	for each $t_{ij} \in D_i$
5	insert $Synonym(t_{ij})$ into D_i' ;
6	insert $Antonym(t_{ij})$ into D_i' ;
7	end for
8	return D_i' ;
9	end for

Furthermore, we conducted a pilot study to compare the text feature words extracted from app reviews and changelogs. The preliminary results indicate that the reflection of app reviews on app changelogs is often expressed as the synonyms and antonyms of a certain text feature word, rather than using the same terms. Since more text feature words benefit pre-training of the classifier, the size of D_i is recommended to be extended to cover more candidate AC-based augmented words. For this purpose, WordNet in NLTK was applied to the initial dictionary D_i to generate the extended dictionary D_i' for the requirements type i . More specifically, for each text feature word t_{ij} in D_i , all its synonyms and antonyms identified in WordNet were added

to generate D_i' . Algorithm 1 provides the details of constructing an AC-based feature dictionary for each type of requirements. This results in D_i' , which will be used in the next step of AUG-AC to augment the app reviews labeled as requirements type i . In our work, D_i' consists of two parts, i.e. (1) the top 20 text feature words extracted from the type i -labeled app changelogs and (2) all the synonyms and antonyms of these 20 words.

D. Augmenting App Reviews

In this step, we select text feature words in the AC-based feature dictionary created in Sect. III.C, in order to augment app reviews. These augmented app reviews construct the training set of the classifier for app reviews classification.

To achieve a higher accuracy in requirements classification from app reviews, we proposed to augment app reviews with the text feature words derived from those app changelogs whose requirements type is identic with that of the app reviews to be augmented. Specifically, for each type of requirements, we first used Word2Vec in NLTK to calculate the similarity between the AC-based text feature words and the app reviews labeled as the same type. Below, formula (1) was defined to perform the similarity calculation task, where r_{ik} denotes the type i -labeled app review sentence k expressed by a vector $r_{ik} = (t_{ik,1}, t_{ik,2}, \dots, t_{ik,m}, \dots, t_{ik,n})$, $t_{ik,m}$ denotes the m -th feature word in r_{ik} , t_{ij} denotes the AC-based text feature word labeled as requirements type i , $w_{ik,m}$ denotes the weight of the feature word $t_{ik,m}$ (produced by BoW) in the app review sentence r_{ik} , $\text{sim}(t_{ik,m}, t_{ij})$ denotes the similarity between the AC-based feature word t_{ij} and the app review-based feature word $t_{ik,m}$ (calculated by Word2Vec), and n denotes the number of AC-based feature words to be augmented to the app review sentences.

$$\text{Sim}(r_{ik}, t_{ij}) = \frac{\sum_{m=1}^n (w_{ik,m} * \text{sim}(t_{ik,m}, t_{ij}))}{\sum_{m=1}^n w_{ik,m}} \quad (1)$$

Considering each app review sentence r_{ik} , the similarity between r_{ik} and t_{ij} , i.e. the value of $\text{Sim}(r_{ik}, t_{ij})$, will be ranked. As a result, the top n AC-based feature words will be added to the end of this review sentence as the semantic augment. This means that n can be treated as the length of AC-based augment for app reviews. Algorithm 2 describes how to augment app review sentences with AC-based text feature words extracted in Sect. III.B and generated in Sect. III.C. Note that in our work, the value of n is pre-specified and fixed for augmenting app reviews. How much length of AC-based augments, i.e. the number of feature words augmented to app reviews, could bring more accurate prediction of app review classification will be discussed in Sect. IV.

Algorithm 2: Generating AC-Augmented App Reviews

Input:	AR_i – app reviews labeled as requirements type i n – the length of AC-based augment
Output:	Aug_AR_i – augmented AR_i by adding n words
1	for each requirements type i
2	for each $r_{ik} \in AR_i$
3	for each word $t_{ij} \in D_i'$
4	calculate $\text{Sim}(r_{ik}, t_{ij})$;
5	end for
6	sort $t_{ij} \in D_i'$ by $\text{Sim}(r_{ik}, t_{ij})$ in descending order;
7	add the first n t_{ij} to r_{ik} to produce new_r_{ik} ;

8	insert new_r_{ik} into Aug_AR_i ;
9	Return Aug_AR_i ;
10	end for
11	end for

E. Classifier Training and Evaluation

According to the experimental results in [1,9,13], Naïve Bayes has been reported to outperform other machine learning algorithms in the automatic classification of app reviews. Therefore, this work adopted Naïve Bayes as the classification technique to categorize FR and NFRs from app reviews. To evaluate the performance of Naïve Bayes, 10-fold cross validation was applied to reduce its overfitting in identification and classification of app reviews. In addition, we adopted the standard metrics *Precision*, *Recall* and *F-measure* to evaluate the accuracy of Naïve Bayes on the automatic classification of app reviews.

$$\text{Weighted average } (Precision_i \backslash Recall_i \backslash F - measure_i) = \frac{\sum_{i \in type} (Precision_i \backslash Recall_i \backslash F - measure_i) * Number_i}{\sum_{i \in type} Number_i} \quad (2)$$

More specifically, for each requirements type i , $Precision_i$ is the fraction of the app reviews that are correctly classified as requirements type i , $Recall_i$ is the fraction of the app reviews of requirements type i that are correctly classified as type i , and $F - measure_i$ is the harmonic average of the precision and recall. Furthermore, we introduced weighted average precision, recall and F-measure (see formula (2) below) to evaluate the accuracy of classifying app reviews into categories of each requirements type. In formula (2), $Number_i$ denotes the number of app review sentences labeled as requirements type i in the test set of Naïve Bayes.

IV. RESULTS

This section reports the results of our experimental study and compares the accuracy of Naïve Bayes in the automatic classification of requirements from app reviews. The Naïve Bayes algorithm was programmed with Python. All the experiments were conducted on a 2.50GHz Core i5 CPU with 8GB RAM under Windows 10.

A. Impact of Length of Augments on App Reviews Classification

As mentioned in Sect. III.C, the length of AC-based augments depends on the number of AC-based text feature words that added to the specified app reviews. Figure 2 shows the precision, recall and F-measure of the automatic classification of augmented app reviews with increasing number of AC-based augmented words (from 5 to 70 words) with an interval of 5 words, by applying Naïve Bayes. Note that in these experiments, the maximum length of AC-based augments is set as 70 words. The reasons are: (1) in [9], Lu and Peng have reported that augments with 1.9 times of the length of an app review sentence leaded to the best results in app reviews classification; and (2) in our dataset, the longest app review sentence has 37 words, and 70 words are around 1.9 times of the maximum length of included app review sentences.

As observed in Figure 2, F-measure is growing rapidly when less than 35 AC-based feature words are augmented to app reviews. There are two peaks when the length of augments is 35

and 45 words respectively. When the number of added AC-based feature words is greater than 50, the value of F-measure decreased or fluctuated within a narrow range.

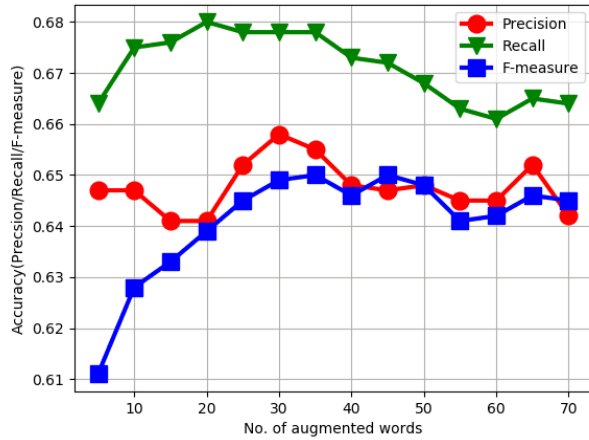


Figure 2. Accuracy of classifying augmented app reviews with varying number of AC-based text feature words.

Furthermore, Table I zooms in the Precision, Recall and F-measure for classifying each type of requirements in app reviews, by employing AC-based augments with different lengths. Typically, the experiments run on the app reviews augmented by 6 (i.e. the average length of app reviews in our dataset) and 37 (i.e. the maximum length of app reviews in our dataset) AC-based text feature words. The results were listed in the columns for precision, recall and F-measure in Table I. We found that for two types or requirements – *Reliability* and *Other*, the accuracy of app reviews classification seldom differs in the lengths of augments for app reviews. Whereas, for *Usability*, *Portability*, *Performance* and *FR* typed app reviews, the longer AC-based augment leads to much higher accuracy of classifying app reviews. Specifically, we analyzed the influence of the proportion of each type of requirements identified in app reviews or changelogs on the accuracy of classifying app reviews with different length of AC-based augments.

As shown in Figure 3(a), for two types of requirements – *Usability* and *FR*, the higher proportion of app changelogs labeled as these two requirements types leads to more accurate identification and classification of these two types of

requirements in app reviews. Similarly, for the other two requirements types – *Portability* and *Performance*, the lower proportion of app changelogs resulted in a lower accuracy of classifying app reviews labeled as these two types. Whereas, it is surprising to find that lower proportion of app changelogs typed as *Reliability* and *Other* produced the two highest accuracy of classifying these two types of requirements in app reviews. Regarding the proportion of six specified types of requirements in app reviews, Figure 3(b) indicates that the higher (lower) accuracy of classifying app reviews labeled as a certain requirements type usually responds to the higher (lower) proportion of app reviews labeled as this type of requirements.

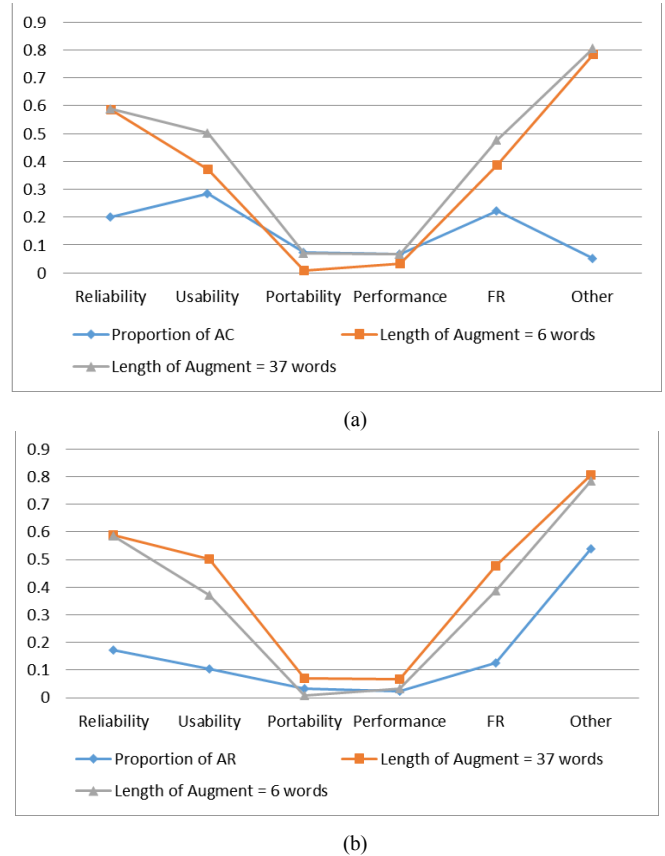


Figure 3. Influence of the proportion of each type in (a) app changelogs and (b) app reviews on the F-measure of classifying augmented app reviews.

TABLE I. PROPORTION OF APP REVIEWS/CHANGELOGS AND ACCURACY OF REQUIREMENTS CLASSIFICATION IN AUGMENTED APP REVIEWS (AC = APP CHANGELOGS, AR = APP REVIEWS)

Type	Proportion of AR	Proportion of AC	Length of Augment = 6 words			Length of Augment = 37 words		
			Precision	Recall	F-measure	Precision	Recall	F-measure
Reliability	0.172	0.199	0.641	0.544	0.586	0.586	0.588	0.587
Usability	0.104	0.285	0.845	0.239	0.370	0.656	0.408	0.502
Portability	0.034	0.073	0.100	0.004	0.007	0.410	0.040	0.071
Performance	0.025	0.068	0.200	0.018	0.034	0.400	0.037	0.068
FR	0.126	0.222	0.582	0.289	0.386	0.468	0.486	0.476
Other	0.538	0.053	0.672	0.951	0.785	0.750	0.873	0.807
Weighted average			0.641	0.665	0.611	0.656	0.678	0.652

B. Comparison with AUG-BoW

Similar to AUG-BoW in [9], our proposed AUG-AC also aims at augmenting app reviews for more accurate classification of app reviews. As we mentioned in Sect. II, AUG-AC differs in employing official app changelogs to augment app reviews for the identification and classification of requirements in app reviews. To compare and evaluate the performance of these two methods, we conducted a series of experiments varying in the methods for generating augments and the length of augments. As listed in the first row of Table II, we repeated AUG-BoW in sampled app reviews in our dataset; while in the second to the fourth row of Table II, the experiments evaluated AUG-AC in the cases that the lengths of augments were the average length of app reviews (i.e. 6.15 words), 1.9 times of this average length, and the maximum length of included app reviews (i.e. 37 words) respectively.

TABLE II. RESULTS ON CLASSIFYING APP REVIEWS AUGMENTED WITH DIFFERENT TECHNIQUES.

Techniques	Length of augment	Precision	Recall	F-measure
AUG-BoW [9]	$1.9 \times \text{length of an app review [9]}$	0.651	0.642	0.569
AUG-AC	6 words	0.646	0.666	0.610
AUG-AC	15 words	0.650	0.674	0.631
AUG-AC	37 words	0.656	0.678	0.652

Our experimental results are in Table II. Therein, we observe that once our proposed AUG-AC was applied to generate AC-based augments for app reviews, the accuracy of app reviews classification increases regardless of the length of augments. That is, our proposed AUG-AC outperformed AUG-BoW [9] by employing app changelogs for classifying requirements in app reviews. Furthermore, we compared the time spent on the automatic classification of augmented app reviews when applying AUG-BoW and AUG-AC in our dataset respectively. For this purpose, the time spent in two typical experiments – in the 1st and 4th row of Table II, was calculated. Note that in this work, we only concentrated on the time spent on augmenting app reviews with app changelogs by these two methods. The reason is that both AUG-BoW and AUG-AC adopted Naïve Bayes as the classifier, and these two techniques may take the same time period to classify augmented app reviews. The results are: (1) AUG-BoW took 1315.77 seconds to construct ‘customized’ augments for each app review in the training set of Naïve Bayes, and (2) AUG-AC took 106.64 seconds to complete the 37-word augments for any included app reviews. It was obvious that app reviews classification based on AUG-AC completed much faster than that based on AUG-BoW.

V. DISCUSSION

A. Analysis on the length of AC-based augments

Regarding the length of AC-based augments, we observed that the longer AC-based augments led to a higher accuracy in classifying app reviews. However, the accuracy did not continuously increase by augmenting app reviews with more than around 40 text feature words. The reason could be that in our AC-based feature dictionary, the AC-based feature words to be added to app reviews were ranked according to their topic

relevance with the specified type of requirements. In turn, the top 40 feature words selected for the construction of AC-based augments were the most ‘type-sensitive’ ones, and also enough, to provide much more accurate results in the app reviews classification.

Next, the proportion of each type of requirements in app reviews and changelogs was not always similar, indicating that these two data sources concentrated on different types of requirements. Taking ‘*Usability*’ NFR as an example, its proportion in app changelogs is around 2.7 times of that in app reviews. This finding implies that although apps may be upgraded to fit different types of user requirements, the official changelogs always pay more attention to this type of requirement. The reason could be that for user, changes typed as ‘*Usability*’ were critical for making decision on whether this release is appropriate for their demands. Regarding to the ‘*Other*’ type, the much lower proportion of app changelogs indicates that they have less noise than app reviews.

Furthermore, we found that the rank of the proportion of different types of requirements in app reviews nearly follow the rank of accuracy in classifying these requirements types in app reviews. The reason could be that both the training and test set of Naïve Bayes consisted of app reviews. In contrast, the rank of the proportion of different types of requirements in app changelogs did not always correspond to the rank of accuracy in classifying app reviews. For example, lower proportion of ‘*Reliability*’ and ‘*Other*’ types in app changelogs contributed to more accurate results in app review classification. One reason could be that for these two types of requirements, the AC-based text feature words that were used to augment app reviews are quite similar to the feature vectors of app reviews. The other reason could be that compared with the other four types of requirements, those that employed AC-based text feature words are much more ‘topic sensitive’ to identify app reviews typed as ‘*Reliability*’ and ‘*Other*’. All the findings are positive to our exploration on using app changelogs to classify requirements in app reviews, and encourage further research on this topic.

B. Comparison between AUG-AC and AUG-BoW

Considering the accuracy of app reviews classification, we found that our proposed AUG-AC method outperformed AUG-BoW provided in [9] – the work that inspired our ideas to improve the accuracy of app reviews classification with app changelogs. Compared with AUG-BoW, AUG-AC spent much less time in augmenting app reviews. The main reason is that in AUG-AC, the augments of app reviews were constructed by calculating the similarity between the AC-based feature words of a certain type of requirements and the app reviews labeled as this type. That is, for each type of requirements, 20 feature words extracted from app reviews and their synonyms and antonyms (around 300 words) selected from WordNet are candidates to be calculated and ranked. Whereas, AUG-BoW augmented app reviews by calculating the similarity between extracted text feature words of a certain requirements type and all the app reviews labeled as any type of requirements. These findings can be deemed as the main advantage of our AUG-AC and the main difference between AUG-BoW and AUG-AC.

VI. THREATS TO VALIDITY

We followed the guidelines in [15-16] to evaluate the possible threats to validity of our experimental results.

Construct validity: Our work reused the dataset in [13]. All the app reviews and changelogs in this dataset were analyzed by three coders independently, on the premise that they had a consistent understanding on different types of requirements, especially on NFR types defined in ISO 25010. As indicated in [13], we believe this threat to construct validity is partially mitigated by following the aforementioned labelling process.

Internal validity: There is an internal threat to validity concerning how the proposed AUG-AC and the Naïve Bayes classifier were programmed. We implemented them by Python. The results of this exploratory study may vary if AUG-AC and Naïve Bayes are implemented in other ways, e.g. in Weka. Thus, how to improve the implementation of AUG-AC and Naïve Bayes remains to be studied. Another internal validity threat is that, not all the app changelogs and reviews were collected from the same platform. Especially, the changelogs of WhatsApp were collected from Apple App Store and the app reviews were from Google Play. Different concerns of users in different platforms may lead to the fact that the AC-based text feature words provide insufficient semantics to app reviews, which may further result in inaccurate classification of app reviews. Therefore, more research is needed on app reviews and changelogs from the same platform.

External validity: We investigated the user reviews of three apps and changelogs of 30 apps which span over three categories and two major mobile operating systems. We believe that the threats to external validity are partially alleviated. Due to the time and resource limitation, we did not cover many apps, and we plan cover more categories of apps to increase the external validity of the study results.

VII. CONCLUSIONS AND FUTURE WORK

This work explored to augment app reviews with official app changelogs, in order to improve the accuracy of classifying requirements, including FR and four types of NFRs, in app reviews. For this purpose, AUG-AC was proposed to augment app reviews with not only the text feature extracted from app changelogs but also their synonyms and antonyms generated by Word2Vec, in the case that both the employed app changelogs and the app reviews to be augmented are labeled as the same type of requirements. Next, a series of experiments was designed to evaluate the performance of AUG-AC by varying the length of AC-based augments. The experimental results indicate that the AC-based augment of app reviews implemented by AUG-AC can improve the accuracy of classifying requirements in app reviews.

To further evaluate the performance of AUG-AC, our next steps are: (1) re-evaluation of AUG-AC on a balanced dataset by leveraging the proportions of different types of requirements in current dataset; (2) evaluation of AUG-AC with app reviews and changelogs of other apps in other categories of Apple App Store

or other app repositories (e.g. Google Play, other Android app stores, etc.); and (3) validation of AUG-AC by using the dataset labeled by more types of NFRs (e.g. Security).

REFERENCES

- [1] W. Maalej, Z. Kurtanovic, H. Nabil, C. Stanik, "On the automatic classification of app reviews", *Requirements Engineering*, vol. 21, no. 3, pp.311-331, 2016.
- [2] E. Guzman, M. El-Haliby, B. , "Ensemble Methods for App Re-view Classification: An Approach for Software Evolution", in *Proc. of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*, Lincoln, USA, 2015, pp.771-776.
- [3] S. Panichella, A. Di Sorbo, et al., "How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution", in *Proc. of IEEE International Conference on Software Maintenance and Evolution (ICSME'15)*, Bremen, Germany, 2015, pp.281-290.
- [4] P.M. Vu, H.V. Pham, T.T. Nguyen, T. T. Nguyen, "Phrase-based extraction of user opinions in mobile app reviews", in *Proc. of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE'16)*, Singapore, 2016, pp.726-731.
- [5] W. Jiang, H. Ruan, et al, "For User-Driven Software Evolution: Requirements Elicitation Derived from Mining Online Reviews", In *Proceeding of the 18th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD'14)*, Taiwan, China, 2014, pp.584-595.
- [6] N. Chen, J. Lin, S.C.H. et al, "AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace", in *Proc. of the 36th International Conference on Software Engineering (ICSE '14)*, Hyderabad, India, ACM, 2014, pp.767-778.
- [7] C. Gao, J. Zeng, M. R. Lyu and I. King, "Online App Review Analysis for Identifying Emerging Issues", in *Proc. of the 40th International Conference on Software Engineering (ICSE'18)*, Gothenburg, Sweden, ACM, 2018, pp.48-58.
- [8] H. Yang and P. Liang, "Identification and Classification of Requirements from App User Reviews", in *Proc. of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE'15)*, Pittsburgh, USA, 2015, pp.7-12.
- [9] M. Lu and P. Liang, "Automatic Classification of Non-Functional Requirements from Augmented App User Reviews", in *Proc. of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE'17)*, Karlskrona, Sweden, 2017, pp.344-353.
- [10] E.C. Groen, S. Kopczynska, et al., "Users-The Hidden Software Product Quality Experts?", in *Proc. of the 25th International Requirements Engineering Conference (RE'17)*, Lisbon, Portugal, 2017, pp.80-89.
- [11] ISO. ISO/IEC 25010, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. FDIS, 2011.
- [12] R. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, ISBN 978-3-662-43838-1. 2014.
- [13] C. Wang, F. Zhang, P. Liang, et al. "Can App Changelogs Improve Requirements Classification from App Reviews? An Exploratory Study", In *Proc. of the 12th International Symposium on Empirical Software Engineering and Measurement (ESEM'18)*. ACM, Oulu, Finland, 2018, pp.43:1-43:4.
- [14] YZ. Liu, L Liu, HX Liu, and WY Wang, "Analyzing reviews guided by App description for the software development and evolution", *Journal of Software: Evolution and Process*, vol. 30, no. 12, 2018, pp. ,
- [15] Forrest Shull, Janice Singer, and Sjøberg Dag I. K. *Guide to Advanced Empirical Software Engineering*. Springer, 2008.
- [16] Claes Wohlin, Per Runeson, Martin Host, Magnus C. Ohlsson, Regnell Bjorn, and Anders Wesslen. *Experimentation in Software Engineering*. Springer, 2012.

Prudent Practices for Designing Virtual Desktop Experiments

Peiyu Liu, Wenzhi Chen, Zonghui Wang, Lirong Fu

College of Computer Science and Technology

Zhejiang University

Hangzhou, China

{liupeiyu, chenwz, zhwang, fulirong007}@zju.edu.cn

Abstract—Virtual desktop technology aims at accessing a remote desktop by endpoint hardware. Great attention has been increasingly paid to virtual desktop since it can increase the utilization of computing resources and provide more flexible accesses. However, researchers have not yet come up with a comprehensive set of rigorous standards of experimental design and implementation in this field. Therefore, it is difficult to conduct prudent experiments, which is correct, real, and transparent.

In this paper, we assess the experimental evaluations of recently published papers on desktop virtualization. We observe that most works can be further improved, due to the unsuitable experimental environment and the lack of descriptions of experimental settings. In this paper, in order to help researchers, reviewers, and readers, we propose several guidelines for designing correct, real, and transparent desktop virtualization experiment.

Index Terms—Virtual Desktop; Prudent Experiment;

I. INTRODUCTION

Nowadays, with the rapid development of virtualization technology, cloud computing has drawn great attention from both academia and industry fields [1], [2]. The thin client was first proposed in the 1990s, which finally develops rapidly in the form of virtual desktops in the boom of cloud computing [3], [4]. There are many mature solutions in the industry fields [5]. The academic community has also proposed a lot of impressive works [6]. Many desktop virtualization approaches will conduct extensive experiments to verify the performance of the proposed desktop virtualization systems. However, since there is no comprehensive set of rigorous standards of experimental design and implementation in this field, researchers will face numerous pitfalls during the experiment.

In this paper, we investigate issues about the prudent experimental evaluation of desktop virtualization systems. We observe that there are no general experimental standards in this field, which makes it difficult to compare these projects equally. In our previous research work, we notice that it is hard to reproduce the experiments that are conducted in many other approaches. Therefore, we put forward that the existing systems of desktop virtualization could be further improved in a rigor experimental standard. We solemnly declare that we highly respect the existing works. Under the purpose of helping researchers, we point out the current common problems that every relevant researcher, including ourselves,

might encounter. One of these problems is that the description of the experiments in many works is sometimes inadequate. In addition, there are more serious issues that may affect the correctness of the experiments.

Our goal is to establish a set of rigor guidelines for the design, implementation, and description of prudent desktop virtualization experiments. We regard *correctness*, *reality*, and *transparency* as three cornerstones. Based on these cornerstones, we propose guidelines that can help researchers in prudent desktop virtualization experiments. We review existing 17 papers under our guidelines to confirm the validity of these guidelines. Most papers can benefit from our proposed guidelines. Through simple case study, we also validate the existence of some pitfalls encountered in the desktop virtualization experiments.

In summary, our work makes the following important contributions.

- (1) We discover the common pitfalls in desktop virtualization experiments.
- (2) We propose a set of guidelines to help researchers improve the rigor of desktop virtualization experiments.
- (3) We investigate 17 desktop virtualization papers and validate that our guidelines are practical.
- (4) We conduct simple case study to prove that the pitfalls in the desktop virtualization experiments deserve great attention.

II. RELATED WORK

In this section, we summarize existing works related to our research.

Virtual Desktop Infrastructure (VDI): The technology that allows users to run desktop operating systems on virtual machines is known as VDI [6]. VDI manages a virtual desktop, which is a desktop environment in a virtual machine (VM) that runs on a centralized or remote server. A user can access a virtual desktop through a variety of terminals. There are many vendors which provide VDI solutions such as Citrix [5] and VMware [7].

Desktop as a Service (DaaS): DaaS provides the benefit of VDI without the cost and risk of managing physical resources, which allows a user to access desktop applications by any devices in anywhere [8]. Deboosere et al. propose a system

architecture to provide efficient desktop services in a cloud [9], which specifically focuses on mobile users. Therefore, due to resource constraints, virtual desktops are executed remotely. Kim et al. design and implement a desktop virtualization system using lightweight display protocols based on cloud DaaS [10].

Virtual Mobile Infrastructure (VMI): VMI extends the principles that allow VDI to run desktop applications on desktops and mobile devices - only this time, mobile apps are accessed remotely from mobile devices [11]. Su et al. propose vMobiDesk, a prototype system which provides mobile users with remote accesses to virtual mobile desktop such as Android desktop [12].

Prudent Practices for Designing Experiments: The rigor of experiments is very important for all academic papers. Previous researchers have analyzed the experimental rigor in other research fields [13]. However, before our paper, the rigor of desktop virtualization experiments has not been studied.

III. GUIDELINES FOR PRUDENT EXPERIMENT

We propose that the main pitfalls in existing desktop virtualization experiments can be divided into three categories. First of all, reasonable experimental setup and environment are the keys to ensure the *correctness* of the experiment. In addition, only real-world testings can *realistically* demonstrate that the virtual desktop really satisfies the actual needs of users. Finally, a *transparent* description of the experimental details can help the reviewers and readers understand the experimental setup and ensure the reproducibility of the experiment. Based on the above cornerstones, the following content outlines the guidelines of prudent virtual desktop experiments.

A. Correct Setting

1) **Select the appropriate test indicators:** The indicators tested in each desktop virtualization approach are not always the same. It is worthy to point out that the testing of parts of indicators is indispensable. For instance, we observe that many papers do not adequately test response time in virtual desktop experiments. However, for desktop virtualization, the performance of a system is strongly related to user interactions, no matter how appreciative other test results in the experiments are, the lack of response time testing will always be confusing. We emphasize that it is necessary to use appropriate methods to test important indicators.

2) **Comprehensively consider the effects of experimental equipment:** Various devices can be used in the tests of the performance of desktop virtualization systems. Obviously, the equipment used by the researchers is not uniform. It is indeed difficult to require everyone to use the same devices. We will not make such unreasonable demands. However, researchers should realize that different equipment may cause deviations in experimental results. We point out that different devices should be used in the same tests to decrease the experimental bias.

3) **Determine the impact of network settings:** Different network configurations are likely to significantly affect the performance of a virtual desktop. In many existing papers, the authors have configured excellent network connectivity, which is almost impossible in real life. In such a network environment, desktop virtualization systems may perform very well. However, virtual desktops may not perform so well in daily network environments. We propose to configure a variety of network environments for comparison.

4) **Pay attention to the effects of desktop resolutions:** In desktop virtualization systems, desktop data can be encoded with any resolution when it is transmitted. When the same desktop data is transmitted by different resolutions, the amount of the data is different, which will have an impact on the network bandwidth and response time. Therefore, we emphasize that authors must pay attention to the influence of resolutions and evaluate them in their papers.

B. Realistic Tests

1) **Conduct real-world experiments:** Desktop virtualization works need to solve real-life problems, which should be able to work in the real world while satisfying the needs of users. To evaluate the actual performance of a desktop virtualization system, real-world experiments should be conducted. We propose that using the equipment of users to conduct experiments in the real-world working environment is more convincing.

2) **Be cautious about the compatibility:** For various reasons, many papers are evaluated only in a single Operating System (OS) version. However, considering the compatibility, we propose that papers should explain whether the current desktop virtualization system can be applied to other OS versions, or investigate in detail how much work is needed to port the desktop virtualization system to other OS versions.

C. Transparent Description

1) **Detail description of the OS and tools used in the experiment:** Different OS versions and test tools may lead to different experimental results. We insist that the author is obliged to elaborate the OS version and test tools in their paper, such as "Windows 7 64-bit none third-party programs installed", "netperf-2.7.0 released on 21 Jul 2015, download address: <https://github.com/HewlettPackard/netperf/releases>", The detail description can improve the possibility of reproducing experiments by readers.

2) **Explain the reasons for the poor/outstanding performance:** If the virtual desktop does not perform well on a test, we strongly recommend that the author should analyze the possible causes of the poor performance carefully. It is always a respectable practice to propose possible improvement solutions. Even if the desktop virtualization system performs well in the experiments, the author still needs to perform a comprehensive analysis. If the experimental environment, such as the condition of the network, is the reason for the outstanding performance of a desktop virtualization system, then ignore this reason is unfair for other papers.

TABLE I
LIST OF SURVEYED PAPERS CLASSED BY TOPIC. SOME TITLES ARE SHORTEN WITH [...].

#	Authors	Title	Venue
VDI			
1	Baratto et al. [14]	MobiDesk: Mobile Virtual Desktop Computing	ACM MobiCom 2004
2	Baratto et al. [6]	THINC: A Virtual Display Architecture for Thin-Client Computing	ACM SOSP 2005
3	Kibe et al. [15]	The Evaluations of Desktop as a Service in an Educational Cloud	IEEE NBIS 2012
4	Alexander et al. [16]	Building a Cloud Based Systems Lab	ACM SIGITE 2012
5	Darabont et al. [17]	Performance Analysis of Remote Desktop Virtualization based [...]	MACRo 2015
6	Kim et al. [10]	Cloud-based Virtual Desktop Service Using Lightweight [...]	IEEE ICOIN 2016
7	Uehara et al. [18]	Performance Evaluations of LXC based Educational Cloud in a [...]	IEEE WAINA 2017
8	Triyason et al. [19]	The impact of screen size toward QoE of cloud-based virtual desktop	Elsevier PCOCEDIA 2017
DaaS			
9	Beaty et al. [8]	Desktop to Cloud Transformation Planning	IEEE ISPDC 2009
10	Cristofaro et al. [20]	Virtual Distro Dispatcher: a light-weight Desktop-as-a-Service [...]	Springer CLOUD 2009
11	Lai et al. [21]	A Service Based Lightweight Desktop Virtualization System	IEEE ICSS 2010
12	Calyam et al. [22]	Utility-directed resource allocation in virtual desktop clouds	Elsevier COMNET 2011
13	Deboosere et al. [9]	Cloud-based Desktop Services for Thin Clients	IEEE INTERNET COMPUT 2012
VMI			
14	Hung et al. [11]	Executing mobile applications on the cloud: Framework and issues	Elsevier COMPUT MATH APPL 2012
15	Nguyen et al. [23]	An Efficient Video Hooking in Androidx86 to Reduce Server [...]	Springer CUTE 2014
16	Su et al. [12]	vMobiDesk: Desktop Virtualization for Mobile Operating System	IEEE HPCC 2017
17	Wang et al. [24]	FUSION: A Unified Application Model for Virtual Mobile [...]	IEEE DSC 2017

TABLE II
CRITERIONS FROM THE CORRESPONDING GUIDELINES. '***': MUST COMPLY. '**': SHOULD BE FOLLOWED. '*': GOOD TO MEET.

Criteria	Guidelines	Rating	Explication
Correct Setting			
Response time test	2.1.(a)	***	Perform response time test in a suitable manner
Bandwidth evaluation	2.1.(a)	***	Test bandwidth in an appropriate way
Frame rate test	2.1.(a)	***	Test the frame rate in a reasonable way
Different devices	2.1.(b)	**	Use different client devices for multiple sets of tests
Diverse network connectivity	2.1.(c)	***	Configure a variety of network environments for comparison
Different resolutions	2.1.(d)	***	Test virtual desktop performance under different resolutions
Realistic Test			
real-world experiments	2.2.(a)	**	Conduct experiments in the real users' work environment
Real users	2.2.(a)	***	Ask real users to experience the system and measured users' ratings.
Multiple OSes	2.2.(b)	*	Conduct experiment with different server OS versions
Transparent Description			
Introduction of OS version	2.3.(a)	***	Detaile the OS version
Introduction of test tools	2.3.(a)	***	Describe the selected test tool in detail
Interpretation of poor performance	2.3.(b)	***	Analyze the possible causes of the poor performance carefully
Analysis of good performance	2.3.(b)	***	Conduct in-depth analysis of good performance
Improvement solutions	2.3.(b)	*	Propose possible improvement solutions

IV. ASSESSMENT OF GUIDELINES

In this section, we elaborate the assessment of the guidelines presented in previous sections. The assessment method we use is to extract criterions from the guidelines and then apply the criterions to 17 recent papers listed in Table I for analysis. Through this method, we validate the practical value of the guidelines and obtain some observations.

A. Process of Assessment

In order to assess our guidelines, we extract more specific criterions from the corresponding guidelines. Table II shows the criterions we proposed. For researchers who study on desktop virtualization, every criterion we define can be judged directly with the corresponding papers. As shown in Table II, we divide all the criterions into three levels. Among them, level '***' indicates that the experiments in a rigorous paper

TABLE III
OVERVIEW AND BRIEF DESCRIPTION OF OBSERVATIONS.

Criteria	Rating	Yes	Weak	Description
Correct Setting				
Response time test	***	8 (47.0%)	2 (11.8%)	Only about half of the papers performed response time test in a suitable manner. In addition, there are two papers that measured response time, but their conclusions are too sloppy. None of the remaining articles measured response time.
Bandwidth evaluation	***	9 (52.9%)	0 (0%)	About half of the papers do not measure bandwidth consumption at all.
Frame rate test	***	3 (17.6%)	0 (0%)	Almost all papers have not evaluated the FPS of virtual desktops.
Diverse network connectivity	***	5 (29.4%)	0 (0%)	Only five papers configured a variety of network environments for comparison.
Different resolutions	***	6 (35.3%)	0 (0%)	Less than half of papers tested the performance of virtual desktops at different resolutions.
Different devices	**	2 (11.8%)	0 (0%)	Most papers do not mention support for multiple client devices. In this case, client compatibility cannot be evaluated.
Realistic Test				
Real users	***	2 (11.8%)	0 (0%)	Few researchers have invited real users to participate in the evaluation of desktop virtualization systems. In other words, basically only the rigid numerical results are provided.
real-world experiments	**	3 (17.6%)	1 (5.9%)	A majority of papers do not conducted real-world experiment. Only two papers met this criterion. In addition, there was a paper that used tools to simulate different user environments.
Multiple OSes	*	1 (5.9%)	0 (0%)	In only one case the authors conducted experiment with different server OS versions.
Transparent Description				
Introduction of OS version	***	10 (1%)	0 (0%)	Seven papers do not describe the operating system version of the virtual desktop. This puts those who wish to reproduce the experiment into a situation where they have no rules to follow.
Introduction of test tools	***	6 (35.3%)	2 (11.8%)	About half of the papers do not present information about the tool. There are also two papers that just mention the name of the tool but do not detail the version and other information.
Interpretation of poor performance	***	6(35.3%)	1(5.9%)	Only about a third of the papers explain the reasons for the poor performance in the experiment. There is also a paper that only explains a part of the poor performance.
Analysis of good performance	***	3 (17.6%)	2 (11.8%)	The vast majority of the papers just put some numerical results that look very good but do not explain any of the deep reasons behind the excellent results. There are also two papers that only explains a part of the good performance.
Improvement solutions	*	1 (5.9%)	0 (0%)	Only one paper proposes improved solutions after discussing the causes of poor performance.

must comply with these criteria. Level ‘**’ shows that these criteria should be followed, while Level ‘*’ means that it is good to satisfy these criteria.

We leverage each criteria in Table II to evaluate the papers in Table I. Two of our authors conduct a survey for all the papers. Our goal is to investigate the prudence of the experiments through all the available information in the paper. Therefore, in the process of investigating the papers, we follow the rules that only focus on the content of the papers. We do not review the source code or contact the authors of the papers for more details. We use these restrictions because they actually reflect what readers and reviewers face. Reviewers are often not likely to investigate details that the author has omitted. In the case of a double-blind submission, there is simply no way to contact the author. In other words, only the paper itself can be easily accessed by reviewers and readers. It is the author’s obligation to clarify the details in the paper.

B. Observations

Table III lists the statistics results of all the surveyed papers. *Yes* refers to papers adhere to the guideline.

1) Correct Setting: About 47% of the papers conduct experiments to test response time. Two papers take how users feel as a performance measure, which is not suitable. Similarly, half of the papers have bandwidth evaluation. For the frame rate test, more than 80% of the approaches ignore the test. No more than 50% of the papers use different devices to evaluate the proposed systems. Only 27.7% of the papers test different network connectivities. 30% of the works conduct the experiment of different resolutions. We can see that most papers do not include enough correct settings.

2) Realistic Test: The survey indicates that few papers conduct real-world experiments. From the table, we observe that 80% of the papers lack real-world experiments and only one

TABLE IV
SETTINGS FOR OUR EXPERIMENTAL PLATFORM.

Server Host	Processor	3.40GHz Intel Core i7-6700
	RAM	8 GB
	OS	Ubuntu 16.04
	Kernel	Linux 4.4.0
Server Guest	Virtualization tool	VirtualBox 5.2.10
	RAM	1 GB of RAM
	OS	Android-x86-5.1.1
Client Device	Hardware	Google Nexus 9
	OS	Android-6.0
	RAM	2GB
Network	Hardware	NETGEAR R8000
	Frequency	2.4GHz/5GHz
	Speed	1000MB

paper conduct experiments with different OS versions. Few researchers (about 11%) have invited real users to participate in the evaluation of desktop virtualization systems.

3) Transparent Description: Half of the papers do not introduce the OS version that they leverage. 50.5% of the papers do not describe the testing tools. Only half of the papers mention network connectivity. Consequently, in the majority of the cases, readers fail to figure out the experiment setup adequately, nor can repeat the experiments. Meanwhile, we find that more than 90% of the papers incompletely describe experimental results. Only two papers offer improvement solutions to address low-performance problems.

V. CASE STUDY

In order to explain the problems caused by the violation of the guidelines intuitively, we design experiments to prove that the aforementioned pitfalls will affect the evaluation results of a desktop virtualization system.

Specifically, we analyze two experiments related to two criteria: (1) the influence of resolution on experimental results, and (2) the impact of network connectivity. The experiments of the remaining criteria will be updated to *arc.zju.edu.cn*. Based on our previous experience in desktop virtualization, we believe that all the factors in the guidelines will affect the experimental results. The desktop virtualization system used in our experiment is vMobiDesk, which is a relatively new VMI framework [12].

A. Experimental Setup

As shown in Table IV, all tests are performed on a server machine with a 3.40GHz Intel Core i7-6700 processor and 8 GB RAM. The server machine runs Ubuntu 16.04 with Linux 4.4.0 kernel. The guest OS is Android-x86-5.1.1, which runs in a virtual machine created by VirtualBox. The tested client device is Google Nexus 9 running on Android-6.0 OS. A 100 Mbps, 1 ms latency LAN network is utilized to construct local 2.4GHz and 5GHz Wifi communication network between the mobile device and the server.

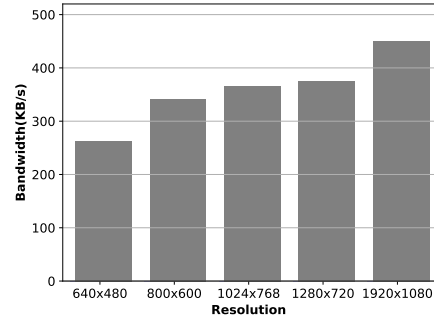


Fig. 1. Bandwidth consumption with different resolutions.

B. The Influence of Resolution

We measure the bandwidth consumption with different resolutions during browsing web pages by vMobiDesk with *iftop-0.17* (released on 12 Feb 2006, download address For <http://www.ex-parrot.com/~pdw/iftop/download/iftop-0.17.tar.gz>). This experiment uses a 2.4GHz wireless network. The results are shown in Table 1. The higher the resolution, the more data of the virtual desktop system needs to be transmitted. Therefore, the higher the bandwidth consumption is required. When the resolution is 640 x 480, the bandwidth consumption is only 262 KB/s. In contrast, it increases to 449 KB/s when the resolution is 1920 x 1080. The increment is up to 71.4%. Without proper handling, this huge consumption will obviously affect the accuracy of an experiment. For instance, a paper ultimately concludes that the desktop virtualization system takes up quite low bandwidth without claiming the use of 640 x 480 resolution for the experiment, which will lead to misunderstandings since the users still consider that the bandwidth consumption remains so small even with a higher resolution.

C. The Impact of Network Connectivity

We further measure the response time under different network connectivity. We conduct experiments in the wireless network environments of 2.4 GHz and 5 GHz, respectively. The average response time is evaluated by several operations such as opening an application, typing several words in a document and returning back to the home screen, etc.

One obvious result is that the response time under 5GHz Wifi is shorter than that under 2.4GHz Wifi. Regardless of the resolution, this difference caused by the frequency of the wireless network always exists. Taking resolution 1024 x 768 as an example, the response time under 5GHz wifi is 410 ms. However, the response time under 2.4GHz wifi rises up to 680 ms. Although this result is not as shocking as the previous experiments shown, 30% of time increment is also large enough to affect the accuracy of an experiment. Leveraging a 5GHz network in an experiment without elaboration will lead to a misunderstanding of the excellent performance.

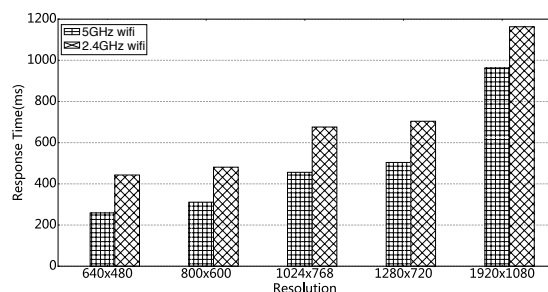


Fig. 2. Response time under different network connectivities.

VI. DISCUSSION

As shown in Section V, improper design of experiments, ambiguous experimental descriptions, and other pitfalls will cause deviations in the assessment of virtual desktop and even lead to misunderstanding. We insist that it is the author's responsibility to ensure the *correctness*, *reality*, and *transparency* of their papers. Surprisingly and disappointingly, our research shows that considering rigorousness, the majority of experiments in desktop virtualization papers are subject to improvement. To reiterate, we greatly respect the researchers, including the authors of the papers we surveyed, and the results of their work. However, it turns out that we have not yet come up with a comprehensive set of rigorous standards of experimental design and implementation in the desktop virtualization field. We believe that a reasonable set of guidelines will help everyone greatly improve their efficiency of work and the quality of the paper. We hope our paper can bring some help to reviewers, authors, and readers. This is the initial motivation for us to carry out this work, and it is also our ultimate goal.

VII. CONCLUSION

In this paper, We summarize pitfalls that are often encountered when conducting desktop virtualization experiments. Based on this, we further propose guidelines that help researchers design and implement prudent experiments in desktop virtualization systems. We extract specific criterions from the corresponding guidelines and leverage each of the guideline criteria to evaluate the papers we selected. Our survey results validate that many papers can be improved. Using our proposed guidelines will help improve the quality of the papers in the experimental part. Finally, we conduct experiments and demonstrate the impact of some pitfalls succinctly.

VIII. ACKNOWLEDGEMENT

We appreciate the anonymous reviewers for their helpful suggestions. Our paper is partly based on the work supported by the National Key R&D Program of China (No. 2016YFB0800201).

REFERENCES

[1] B. Hayes, "Cloud computing," *Communications of the Acm*, vol. 51, no. 7, pp. 9–11, 2008.

[2] P. K. Sahoo, C. K. Dehury, and B. Veeravalli, "Lvrm: On the design of efficient link based virtual resource management algorithm for cloud platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2018.

[3] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, vol. 2, no. 1, pp. 33–38, Jan. 1998.

[4] S. Haripriya., R. Indumathi., and M. S. Manikandan, "Virtual network connection using mobile phones," *Compusoft International Journal of Advanced Computer Technology*, vol. 3, no. 6, 2014.

[5] "HDX Technologies Optimize User Experience - Citrix," <https://www.citrix.com/products/xenapp-xendesktop/hdx-technologies.html>.

[6] R. A. Baratto, L. N. Kim, and J. Nieh, "THINC: A Virtual Display Architecture for Thin-client Computing," in *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, ser. SOSP '05. ACM, 2005, pp. 277–290.

[7] "Horizon 7 | Virtual Desktop Infrastructure | VDI | VMware," <https://www.vmware.com/products/horizon.html>.

[8] K. Beatty, A. Kochut, and H. Shaikh, "Desktop to cloud transformation planning," in *IEEE International Symposium on Parallel and distributed Processing*, 2009, pp. 1–8.

[9] L. Deboosere, B. Vankeirsbilck, P. Simoens, F. D. Turck, B. Dhoedt, and P. Demeester, "Cloud-based desktop services for thin clients," *IEEE Internet Computing*, vol. 16, no. 6, pp. 60–67, 2012.

[10] S. Kim, J. Choi, S. Kim, and H. Kim, "Cloud-based virtual desktop service using lightweight network display protocol," in *International Conference on Information NETWORKING*, 2016, pp. 244–248.

[11] S. H. Hung, C. S. Shih, J. P. Shieh, C. P. Lee, and Y. H. Huang, "Executing mobile applications on the cloud: Framework and issues," *Computers and Mathematics with Applications*, vol. 63, no. 2, pp. 573–587, 2012.

[12] K. Su, P. Jiang, Z. Wang, and W. Chen, "vmobidesk: Desktop virtualization for mobile operating system," in *IEEE International Conference on High PERFORMANCE Computing and Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science and Systems*, 2017, pp. 945–950.

[13] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. V. Steen, "Prudent practices for designing malware experiments: Status quo and outlook," in *Security and Privacy*, 2012.

[14] R. A. Baratto, S. Potter, G. Su, and J. Nieh, "Mobidesk:mobile virtual desktop computing," in *International Conference on Mobile Computing and NETWORKING*, 2004, pp. 1–15.

[15] S. Kibe, T. Koyama, and M. Uehara, "The evaluations of desktop as a service in an educational cloud," in *International Conference on Network-Based Information Systems*, 2012, pp. 621–626.

[16] J. Alexander, A. Dick, J. Hacker, D. Hicks, and M. Stockman, "Building a cloud based systems lab," in *Conference on Information Technology Education*, 2012, pp. 151–154.

[17] Á. Darabont, K. J. Kiss, and J. Domokos, "Performance analysis of remote desktop virtualization based on hyper-v versus remote desktop services," *Macro*, vol. 1, no. 1, pp. 125–134, 2015.

[18] M. Uehara, "Performance evaluations of lxc based educational cloud in a bare metal server," in *International Conference on Advanced Information NETWORKING and Applications Workshops*, 2017, pp. 415–420.

[19] T. Triyason, W. Krathu, T. Triyason, and W. Krathu, "The impact of screen size toward qoe of cloud-based virtual desktop," *Procedia Computer Science*, vol. 111, pp. 203–208, 2017.

[20] S. Cristofaro, F. Bertini, D. Lamanna, and R. Baldoni, "Virtual distro dispatcher: A light-weight desktop-as-a-service solution," in *International Conference on Cloud Computing*, 2009, pp. 247–260.

[21] G. Lai, H. Song, and X. Lin, "A service based lightweight desktop virtualization system," in *International Conference on Service Sciences*, 2010, pp. 277–282.

[22] P. Calyam, R. Patali, A. Berryman, A. M. Lai, and R. Ramnath, "Utility-directed resource allocation in virtual desktop clouds," *Computer Networks*, vol. 55, no. 18, pp. 4112–4130, 2011.

[23] T. D. Nguyen, C. T. Huynh, H. W. Lee, and E. N. Huh, "An efficient video hooking in androidx86 to reduce server overhead in virtual desktop infrastructure," in *Ubiquitous Information Technologies and Applications*. Springer Berlin Heidelberg, 2014, pp. 107–114.

[24] C.-M. Wang, Y.-S. Wu, and H.-H. Chung, "Fusion: A unified application model for virtual mobile infrastructure," in *IEEE Conference on Dependable and Secure Computing*, 2017.

CrowDevBot: A Task-Oriented Conversational Bot for Software Crowdsourcing Platform

Zeyu Ni, Zhangyuan Meng, Junming Cao, Beijun Shen*, Yuting Chen

School of Electronic Information and Electrical Engineering

Shanghai Jiao Tong University, Shanghai, China

{sfordj.ni, m602389789, junmingcao, bjshen, chenyt}@sjtu.edu.cn

Abstract—With the trends of developing software on the Internet, many software crowdsourcing platforms are emerging. They attract a lot of developers to bid for crowdsourced projects and develop software systems collaboratively. In this paper, we present CrowDevBot, a *task-oriented conversational bot for software crowdsourcing platform*, that aims to assist online users in completing crowdsourcing-related tasks in a more natural manner. The key idea of CrowDevBot is to: (1) combine a rule-based method and an SVM-NaiveBayes-C4.5 integrated learning method to discover users' intention; (2) employ an integrated CRF (conditional random field) method with novel features to improve the performance of slot filling; and (3) leverage a software service knowledge base to unify entity names and predefine the key slots of user query. We implement CrowDevBot and integrate it into JointForce, an IT software crowdsourcing platform in China. To the best of our knowledge, this is the first time that a task-oriented conversational bot is practically used in software crowdsourcing platform(s). We evaluated our approach on real data set from JointForce. The results show that our intention detecting method achieves F1-score of 87% on the limited training data. For the slot filling, the F1-score of our integrated CRF model reaches 82%, 8% higher than that of the normal CRF model.

Index Terms—Task-oriented conversational bot, software crowdsourcing platform, integrated statistical learning, user intention understanding

I. INTRODUCTION

Software crowdsourcing [1] is a new software development paradigm. In a crowdsourcing process, project requesters and software developers need to complete many tasks, such as recommending qualified developers, searching and bidding projects, querying project progresses, evaluating service qualities, etc. This paper presents a *task-oriented conversational bot* (CrowDevBot) for software crowdsourcing platform. A bot can assist users in completing their crowdsourcing-related tasks through conversation, rather than tedious mouse clicks. Thus it supplements crowdsourcing platforms with strong flexibility. Figure 1 demonstrates how such a bot helps an engineer to complete his/her task.

Though developing bots can benefit crowdsourcing platforms in completing software crowdsourcing tasks in a more friendly manner, it still faces three challenges:

Challenge 1. *How is the cold start problem solved by a task-oriented conversational bot?* Different from online-shopping

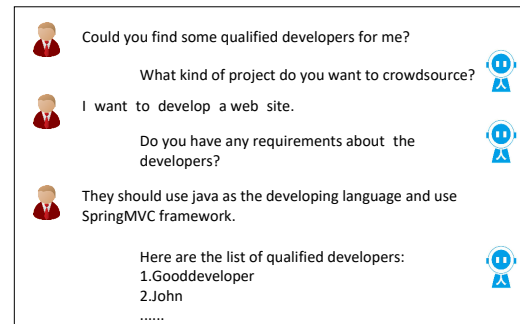


Fig. 1: An example of using CrowDevBot in crowdsourcing

websites, user interactions on software crowdsourcing platform is much less, which brings the cold-start problem. Statistical learning models or deep learning models may also not work well due to limited data.

To solve the cold start problem, several rule-based methods have been proposed for natural language understanding [2] [3] [4]. However, rule-based methods usually have following shortcomings: (1) The accuracy of understanding users' intention relies on the quality of rules. (2) User queries are usually flexible while a pre-defined rule set may not be complete.

Challenge 2. *How is the performance of CRF model improved for user intention understanding?*

CRF (Conditional Random Field) is a widely used model for solving the slot filling problem of user intention understanding. Researches [5] [6] show that CRF can be a general model, while many new deep learning models are not [7] [8] [9].

Due to limited training data, the normal CRF is still unsatisfactory. To improve the slot filling performance of CRF model, there are two key issues: (1) how to define effective features according to the characteristics of software crowdsourcing domain; and (2) how to design an improvement strategy to achieve a more robust CRF model.

Challenge 3. *How can the software service knowledge base be utilized to enhance the capability of the bot?*

Priori knowledge on software crowdsourcing can definitely enhance the capability of CrowDevBot. Inspired by the work of Zhao et al. [10], we build a knowledge base to represent software service knowledge. As far as we know, this is the first time that a task-oriented conversational bot is practically used

*Corresponding author

DOI reference number: 10.18293/SEKE2019-068

in software crowdsourcing scenario. So it will be challenging for us to design and leverage the software service knowledge base in CrowDevBot, to improve its performance.

To address these challenges, we propose a novel approach to developing CrowDevBot. CrowDevBot consists of five key components: *user intention detecting*, *slot filling*, *dialog management*, *task execution*, and *answer generation*. We combine the rule-based method and a SVM-NaiveBayes-C4.5 integrated learning method to deal with the cold start problem when detecting user intentions. CrowDevBot adaptively sets weights for these two methods. For filling slots, we propose an integrated CRF model with novel features, including syntactic and semantic features. We also leverage a software service knowledge base to predefine the key slots of software services and normalize their entity names. So far, we have integrated CrowDevBot into JointForce, one of the biggest IT software crowdsourcing platforms in China. To the best of our knowledge, this is the first time that a task-oriented conversational bot is used in software crowdsourcing platform.

II. RELATED WORK

Storey and Zagalsky [11] propose that bots can act as “conduits between users and services, typically through a conversational UI”. Roughly bots can be divided into two categories [12]: chat-oriented bots and task-oriented bots. In recent years some task-oriented conversational bots have been built both in industry (such as Apple’s Siri, Microsoft’s Cortana) and in academia (such as those proposed by Zhou et al. [10], and Wen et al. [13]). Bots are also rapidly becoming a general interface for software services communication [14]. However, to our best knowledge, few task-oriented conversational bots have been developed for software crowdsourcing platforms.

User Intention Detecting. User Intention Detecting is an important part of task-oriented bots, which can be seen as a classification problem. Particularly, more and more deep models are proposed for this purpose. Kim Y et al. [15] first apply the CNN (convolution neural network) model to solving these problems. Xu et al. [16] use RNN (recurrent neural network) to detect user intentions. Compared to the CNN model, RNN makes use of the sequence information, making it fit for natural language problem. However, no matter how novel the network structure, the performance is poor when data is limited. It requires bots to handle this problem.

Slot Filling. After user intention is detected, the bot needs to parse the user input and recognize the predefined key slots (words to be filled in a sentence). It can be taken as a named entity recognition problem. McCallum [5] et al. use the traditional CRF to solve this problem. Though this model works, it is unable to handle the OOV problem, i.e., it cannot recognize an entity which do not appear in the training data. Thus more novel features need to be designed. Researches [17] apply deep learning methods on solving this problem. But as mentioned before, the restrictions on data reduce the performance of these models.

Cold-Start Problem. When a task-oriented conversational bot is built, it faces with the potential cold-start problems. The cold-start problem is serious for the software crowdsourcing platform as the user data is limited [18]. To solve the cold-start problem in a universal way, Rieser V et al. [4] focus on developing a structured ontology for parsing utterance from user into predefined semantic slots. Zhao Yan et al. [10] present a general solution to the cold-start problem in online-shopping domain. They use crowdsourcing to label training data, while it is not suitable for our situation as data is limited.

III. APPROACH

A. Approach Overview

Our approach overview is shown in Figure 2. The task-oriented conversational bot for software crowdsourcing platform, named CrowDevBot, consists of 5 components.

1) *User Intention Detecting*: Given a user query, CrowDevBot detects the user intentions with a combination method after entity name unification. A rule-based method and an SVM-NaiveBayes-C4.5 integrated learning method is combined by the following strategy: the rule based method is mainly used in the CrowDevBot’s startup process; with data increases, the SVM-NaiveBayes-C4.5 integrated learning method becomes more effective, and thus its weight gets increased.

2) *Slot Filling*: This component gathers the key information (called “slot”) from user queries. We design several key features (including transition feature, start and end features, word and syntactic features), and employ integrated CRF model to fill slots together with probability distribution information to achieve better precision and robustness.

3) *Dialogue Management*: In this component, the interaction process between user and CrowDevBot is managed. CrowDevBot uses a FSM (Finite State Machine) to track the dialogue states. When some slots are missing, it launches new questions to ask. We utilize the software service knowledge base (SSKB) to predefine the key slots for software services. When all of the slots are filled, CrowDevBot invokes *task executing component* to execute the task.

4) *Task Executing*: In this component, the user tasks are executed: once CrowDevBot has identified user intention and filled all slots, it will invoke the corresponding APIs of software crowdsourcing platform and return results to CrowDevBot.

5) *Answer Generation*: As all the supporting tasks are clear and well defined, it can be easy to use a template library to generate answers. CrowDevBot uses FSM to determine which template to be used to generate answers or launch new questions.

Next explains the details of the software service knowledge base, user intention detecting and slot filling.

B. Software Service Knowledge Base

To leverage the domain knowledge of software crowdsourcing, we build a software service knowledge base (SSKB). The main data sources of SSKB includes the knowledge

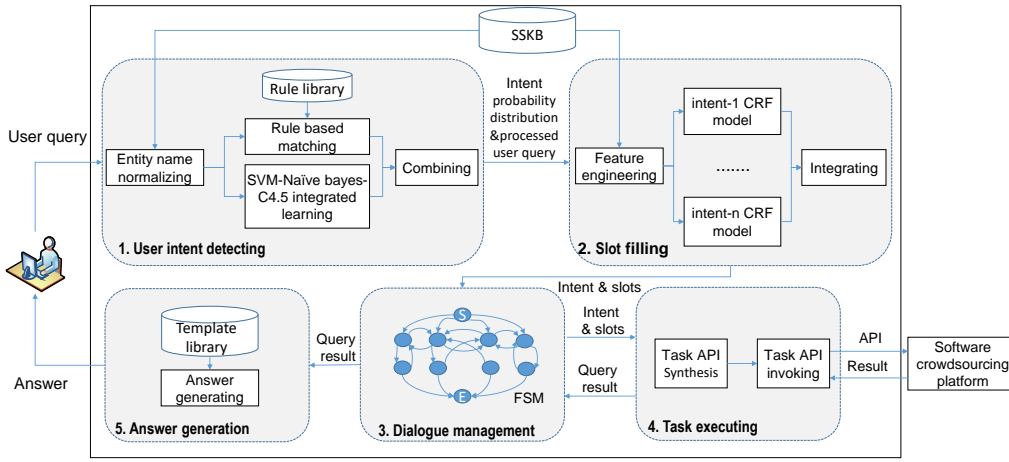


Fig. 2: A general process of CrowDevBot in processing user queries in a software crowdsourcing platform

collected from software crowdsourcing experts, StackOverflow tag synonym system* and Wikipedia†. We apply NLP technology to extract information from these (semi)-structured data, and construct SSKB, which contains category system, technologies, and attributes of software services. For more details, see our project in github‡.

We exploit SSKE to improve the performance of CrowDevBot in two scenarios. The first scenario is entity name unification. In the dialogue with CrowDevBot, users tend to express casually, with various aliases and abbreviations, leading to decreases in precision of intention detecting and slot filling. Thus we build a synonym dictionary using entity synonym attributes in SSKE. During preprocessing, CrowDevBot replaces aliases and abbreviations with standard names.

The second scenario exists in defining slots for software services. For example, in Figure 1, after CrowDevBot receives user input “I want to develop a web site”, it maps the query to the entity “web development” in SSKE. Thus CrowDevBot continuously asks user about the attributes values of “web development” to complement the corresponding slots.

C. User Intention Detecting

User intention detecting can be formulated as a user query classification problem. To solve it, we propose a mixture method, which combines a rule based method and an SVM-NaïveBayes-C4.5 integrated learning method. Weights are dynamically assigned to these two methods. In the startup process, CrowDevBot relies mainly on the rule based method, as it requires less usable training data. After sufficient data is collected, the SVM-NaïveBayes-C4.5 integrated learning method will make more contribution.

1) *Rule-Based Method*: Each kind of intention corresponds to a set of query rules that are used to describe the possible ontology structures of user queries. A rule consists of several

tokens. Each token has 3 attributes: Match_pattern, Weight and Indispensable. Here Match_pattern contains its candidate words, Weight shows the importance of the token to the whole rule, and Indispensable explains the necessities of a token.

The matching algorithm is designed as Algorithm 1 shows.

Algorithm 1 Greedy rule matching algorithm.

Require: A rule (R) and word list (QL) of user query

Ensure: Matching degree (MD) between the rule and query

```

1: for each word or parse  $\in QL$  do
2:   for each unmatched_token  $T \in R$  do
3:     compute the word2vec vector similarity between
        $w$  and each word in  $T.Match\_pattern$ , and record the
       highest  $HS_w$ 
4:     if  $HS > 0.5$  then
5:       mark this token as matched_token and record
       the similarity
6:       goto step 1
7: if all tokens in  $(R.token | R.token.indispensable = 1)$ 
   are matched then
8:    $MD = \frac{\sum_{a \in matched\_token} token_a.weight \times HS_a}{\sum_{b \in R.token} token_b.weight}$ 
9: else
10:   $MD = 0$ 
11: return MD

```

2) *Integrated Learning Method*: CrowDevBot faces with the cold start problem, making deep learning techniques inappropriate to use. Thus we propose an integrated statistical learning method to train the intention classification model. We select SVM, Naive Bayes and C4.5 DT as basic models and combine them following two strategies: Bagging and AdaBoost.

We design three features after manual analysis on these wrong detection cases.

- **Semantic feature.** The feature is produced by POS tagging. The possible values includes noun, adjective, etc.

*<http://stackoverflow.com/tags/synonyms>

†<https://www.wikipedia.org>

‡<https://github.com/SE1405Lab/SSKB>

- N-gram feature. The n-gram feature [19] takes the word sequence-level information to help analyze a sentence. In our method, we take N as 3.
- Word2vec feature. The Word2Vec feature [20] represents words with a low-dimensional vector, which helps understand the semantics of words.

3) *Method Combination*: The results of these two methods above are combined with weights:

$$R = W_1 R_1 + W_2 R_2, \quad (1)$$

where W_1 is the weight of rule-based method, and R_1 is the match degree between user query and each intention; W_2 is the weight of SVM-NaiveBayes-C4.5 method, and R_2 is its probability distribution. The intention with the highest R will be regarded as the final output.

D. Slot Filling

We use CRF [6] as a basic model to fill the slots, as CRF achieves better performance than general models (like HMM [21]). Given the word sequence $X = (x_1, x_2, \dots, x_m)$, CRF computes the conditional probability of a label sequence. It produces a label $y = (y_1, y_2, \dots, y_m)$ sequence to maximize $p(y|x)$.

$$p(y|x) = \frac{1}{z_\lambda(x)} \exp\{\lambda \cdot f(y, x)\} \quad (2)$$

Meanwhile, it is intractable to compute $f(y, x)$. Thus we assume that the value of $f(y, x)$ depends only on the adjacent labels and the formula $p(y|x)$ can be converted into

$$p(y|x) = \frac{1}{z_\lambda(x)} \exp\left\{\sum_{i=1}^{M+1} \lambda \cdot f(y_{i-1}, y_i, x, i)\right\}, \quad (3)$$

where $z_\lambda(x)$ is a normalization factor, and λ the weight vector of feature functions. The training process aims to find a proper λ for the CRF model.

We design several types of features for CRF:

- Transition feature: A transition feature describes the transition between adjacent labels.
- Start feature: A start feature implies that a label happens to be at the start position.
- End feature: An end feature implies that a label happens to be at the end position.
- Word feature: A word feature represents the co-occurrence of a word and a label.
- Syntactic feature: A syntactic feature represents the co-occurrence of a POS tag of the word and a label.
- Semantic feature: Inspired by Li et al.'s work [22], we design the semantic feature using "lexicons", which are clusters of semantically-related word or phrase constructs. A semantic feature describes the co-occurrence of an element in a lexicon L and a label.

We train several CRF models for each intention and integrate them into one mixture model to fill slots of templates. After the slot values are obtained, CrowDevBot uses SSKE to examine whether the key slots are complete and then traces the states using the FSM.

TABLE I: Intention detecting results w.r.t. different models

Model	Precision	Recall	F1-score
Integrated (Bagging)	0.78	0.85	0.80
Integrated (Boosting)	0.83	0.89	0.83
SVM	0.77	0.81	0.78
Naive Bayes	0.65	0.73	0.68
DT(C4.5)	0.75	0.75	0.75
RNN	0.76	0.82	0.78

TABLE II: Intention detecting results with different methods

Method	Precision	Recall	F1-score
Rule Based	0.90	0.61	0.73
Integrated Learning	0.83	0.89	0.83
Combination	0.90	0.85	0.87

IV. EXPERIMENTS

We evaluate our bot on real data from JointForce.

A. Experiment Setup

Three evaluation metrics are selected: Precision, Recall and F-1 score. We have collected 1458 user queries from JointForce[§]. These queries are classified into 6 categories, in each of which we picked up 200 queries with slot information and labeled them manually. We also asked Chinese linguists to design 65 rules such that our rule-based method can be applied.

B. Intention Detecting Experiments

We compared the effectiveness of each single statistical learning model and the integrated model in detecting user intentions. Due to the small amount of our data, we used the BootStrapping method to obtain the training and testing dataset.

The results are shown in Table I. The integrated statistical learning model (with the Boosting training strategy) achieves the highest precision and recall. On the contrary, the deep models are not well supported by a small scale dataset, leading to lower precision and recall.

Next we compared the effectiveness of the rule based method, the integrated statistical learning method and the combination method. The results, as Table II shows, denote that the combination method is slightly better than the statistical learning method. The rule-based method achieves a high precision, indicating the benefits from strictness. The rule matching algorithm implies that, once the user query is matched with one rule, it is much possible to be a true positive.

C. Slot Filling Experiments

We analyzed the feature contributions to slot filling, through an experiment that applies our integrated CRF model with different feature sets. The result is shown in Table III, where iCRF represents our integrated CRF model. It demonstrates that Word Feature (WF), Transition Feature (TF), Semantic Feature (SyF) and Syntactic Feature (SyF) contribute a lot to slot filling, while Start Feature (StF) and End Feature (EF) are less important. Also we can observe that the F1-score of our

[§]<http://www.jfh.com>

TABLE III: Slot filling results with different feature sets

Model	Precision	Recall	F1-score
Normal CRF	0.78	0.71	0.74
iCRF+WF	0.53	0.45	0.49
iCRF+TF	0.38	0.30	0.33
iCRF+EF	0.08	0.04	0.05
iCRF+StF	0.05	0.02	0.03
iCRF+SeF	0.30	0.28	0.29
iCRF+SyF	0.35	0.31	0.33
iCRF+WF+TF	0.63	0.57	0.60
iCRF+WF+TF+EF	0.65	0.58	0.61
iCRF+WF+TF+EF+StF	0.66	0.60	0.63
iCRF+WF+TF+EF+StF+SeF	0.75	0.68	0.71
iCRF+WF+TF+EF+StF+SeF+SyF	0.86	0.78	0.82

TABLE IV: Results of entity name unification

With Entity Name unification	Precision	Recall	F1-score
Yes	0.86	0.78	0.82
No	0.80	0.75	0.77

iCRF model reaches 82%, 8% higher than that of the normal CRF model.

Besides, in order to evaluate the effectiveness of entity name unification using SSKB, we performed slot filling with and without the preprocessing step. The results are shown in Table IV, which indicate that entity name unification preprocessing improves the performance of our integrated CRF method on the slot filling problem. Obviously, the number of words that are out of vocabulary can be significantly reduced through this preprocessing step.

V. CONCLUSION

This paper proposed an approach to building a task-oriented conversational bot (CrowDevBot) for software crowdsourcing platform. Several experiments are conducted to evaluate our approach and CrowDevBot, using the real data from JointForce. Experimental results show that the F1-score of our intention detecting mixture method is 87% under the limited training data, and the F1-score of our integrated CRF model with novel features to fill slots is 82%, up to 8 points higher than normal CRF. Also the user satisfaction ratio of CrowDevBot reaches 87% in average, which indicates that CrowDevBot really helps online users to finish software crowdsourcing tasks in real situation.

As for future work, we will leverage the reinforcement learning technology to improve the performance of our intention detecting and slot filling approach continuously, with better use of the user feedbacks.

ACKNOWLEDGEMENT

This research was sponsored by the National Key Research and Development Program of China (Project No. 2018YFB1003903), National Nature Science Foundation of China (Grant No. 61472242 and 61572312), and Shanghai Municipal Commission of Economy and Informatization (No. 201701052).

REFERENCES

- [1] W. Wu, W. T. Tsai, and W. Li, "Creative software crowdsourcing: From components and algorithm development to project concept formations," *International Journal of Creative Computing*, vol. 1, no. 1, pp. 57–91, 2013.
- [2] N. Gupta, G. Tur, D. Hakkani-Tur, S. Bangalore, G. Riccardi, and M. Gilbert, "The att spoken language understanding system," *IEEE Transactions on Audio Speech and Language Processing*, vol. 14, no. 1, pp. 213–222, 2006.
- [3] R. De Mori, F. Bechet, D. Hakkani-Tur, and M. McTear, "Spoken language understanding," *Signal Processing Magazine IEEE*, vol. 25, no. 3, pp. 50–58, 2008.
- [4] V. Rieser and O. Lemon, *Natural language generation as planning under uncertainty for spoken dialogue systems*. Springer-Verlag, 2010.
- [5] A. McCallum and W. Li, "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons," in *Proc. of the Conference on Computational Natural Language Learning*, 2003, pp. 188–191.
- [6] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Eighteenth International Conference on Machine Learning*, 2001, pp. 282–289.
- [7] F. Deroncourt, Y. L. Ji, and P. Szolovits, "Neuroner: an easy-to-use program for named-entity recognition based on neural networks," 2017.
- [8] M. Habibi, L. Weber, M. Neves, D. L. Wiegandt, and U. Leser, "Deep learning with word embeddings improves biomedical named entity recognition," *Bioinformatics*, vol. 33, no. 14, p. i37, 2017.
- [9] T. H. Pham and P. Le-Hong, "End-to-end recurrent neural network models for vietnamese named entity recognition: Word-level vs. character-level," in *International Conference of the Pacific Association for Computational Linguistics*, 2017, pp. 219–232.
- [10] Z. Yan, N. Duan, P. Chen, M. Zhou, J. Zhou, and Z. Li, "Building task-oriented dialogue systems for online shopping," in *AAAI*, 2017, pp. 4618–4626.
- [11] M. A. Storey and A. Zagalsky, "Disrupting developer productivity one bot at a time," in *ACM Sigsoft International Symposium on Foundations of Software Engineering*, 2016, pp. 928–931.
- [12] P. H. Su, M. Gasic, N. Mrki, L. M. R. Barahona, S. Ultes, D. Vandyke, T. H. Wen, and S. Young, "On-line active reward learning for policy optimisation in spoken dialogue systems," in *Meeting of the Association for Computational Linguistics*, 2016, pp. 2431–2441.
- [13] T.-H. Wen, D. Vandyke, N. Mrksic, M. Gasic, L. M. Rojas-Barahona, P.-H. Su, S. Ultes, and S. Young, "A network-based end-to-end trainable task-oriented dialogue system," *arXiv preprint arXiv:1604.04562*, 2016.
- [14] C. Lebeuf, M. A. Storey, and A. Zagalsky, "Software bots," *IEEE Software*, vol. 35, no. 1, pp. 18–23, 2018.
- [15] Y. Kim, "Convolutional neural networks for sentence classification," *Eprint Arxiv*, 2014.
- [16] P. Xu and R. Sarikaya, "Convolutional neural network based triangular crf for joint intent detection and slot filling," in *Automatic Speech Recognition and Understanding*, 2014, pp. 78–83.
- [17] A. Jaech, L. Heck, and M. Ostendorf, "Domain adaptation of recurrent neural networks for natural language understanding," pp. 690–694, 2016.
- [18] Y. Yang, W. Mo, B. Shen, and Y. Chen, "Cold-start developer recommendation in software crowdsourcing: A topic sampling approach," in *SEKE*, 2017, pp. 376–381.
- [19] W. B. Cavnar, "N-gram based text categorization," in *Proc. Third Symposium on Document Analysis and Information Retrieval*, 1994, pp. 161–175.
- [20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *Computer Science*, 2013.
- [21] Y. Y. Wang, A. Acero, J. Lee, and J. Lee, "Combining statistical and knowledge-based spoken language understanding in conditional models," in *Coling/acl on Main Conference Poster Sessions*, 2006, pp. 882–889.
- [22] X. Li, "Understanding the semantic structure of noun phrase queries," in *Meeting of the Association for Computational Linguistics*, 2010, pp. 1337–1345.

Retrieving Curated Stack Overflow Posts from Project Task Similarities

Glaucia Melo

David R. Cheriton

*School of Computer Science
University of Waterloo
Waterloo, Canada
gmelo@uwaterloo.ca*

Toacy Oliveira

PESC/COPPE

*Universidade Federal
do Rio de Janeiro
Rio de Janeiro, Brazil
toacy@cos.ufrj.br*

Paulo Alencar

David R. Cheriton

*School of Computer Science
University of Waterloo
Waterloo, Canada
palencar@uwaterloo.ca*

Don Cowan

David R. Cheriton

*School of Computer Science
University of Waterloo
Waterloo, Canada
dcowan@uwaterloo.ca*

Abstract—Software development depends on diverse technologies and methods and, as a result, software development teams often need to handle issues in which team members are not experts. To address this lack of expertise, developers typically rely on information obtained from web-based Q&A sites such as Stack Overflow, a popular platform to find solutions to specific technology-related problems. However, access to these Q&A websites is currently not explicitly integrated with software development projects. Therefore, software developers often need to search for solutions to similar and recurring issues multiple times. This lack of integration not only hinders the reuse of the knowledge obtained but also compels developers to perform repeated searches for recurring problems. In this paper, we investigate an approach that explicitly associates project tasks with Stack Overflow posts that have already been curated by developers, and use project task similarities to investigate the possibility to suggest curated Stack Overflow posts. Precision and accuracy were 71.60% and 77.78%, respectively. We also found indications that project task elements such as the process activity, influence accuracy and precision if attempting to reuse curated Stack Overflow posts.

Index Terms—Software engineering, stack overflow, project task, text similarity.

I. INTRODUCTION

Software development is a knowledge-intensive collaborative activity [1]. Currently, development changes with a variety of technologies in use. Therefore, new knowledge must be constantly gathered and software engineers need to engage in tasks that are related to knowledge management, such as learning, capturing, and reusing collaborative knowledge during a software project [2]. During the execution of a software development project, knowledge and expertise from developers are vital for the project to succeed [2].

Software development is usually an integrated system of code editors and debuggers [3] in which developers interact to build a software product. To acquire external support (e.g., code snippets), developers frequently switch between the development environment and browsers [4]. In other words, developers must leave the development environment, reason about relevant and accurate terms for searches, open a browser, verify the results of the search, check if the source is reliable, and only then, transfer the knowledge obtained to the software

[5] [4]. Such activity usually occurs more than once, as software projects are large-scale and iterative. We refer to this effort of tapping into sources of support, reasoning about the help needed and choosing among the vast available content, as *curation*. We use curation inspired by humanities, and which is defined as:

”Select, organize, and present (online content, merchandise, information, etc.), typically using professional or expert knowledge.”

Essential sources of knowledge (information software developers use for support while working) are question and answer (Q&A) websites. A famous Q&A website for software engineers is Stack Overflow (SO) [6] [7] [8] [9]. Although SO is widely used during software development [6], there are still issues about explicitly associating the tasks performed during development and the knowledge obtained from a SO post. The lack of integration of the support often needed by software developers with the development project is identified by researchers as an open issue [10] [11]. Some works have proposed solutions to integrate SO with the software project through text overlap, mainly focusing on issues (bugs, exceptions) [11] [12].

Research indicates that developers’ expertise largely contributes to the success of software projects [2]; however, current approaches to selecting SO posts that use text overlap do not consider developers’ expertise. Integrating curated SO posts with software development projects could help developers avoid performing curation multiple times, redundantly. Additionally, integrating SO posts can convey other benefits, such as keeping relevant information into the project, avoiding searches of the same information, helping less experienced developers know how experts are working and reducing workflow interruptions [4].

In this paper, we investigate the possibility to reuse curated SO posts based on project task similarity by performing an investigation on task contexts and submitting these contexts to a similarity retrieval model. Precision and accuracy (the most common metrics identified among our related works) were collected. Two research questions guide the evaluation of the implemented model: RQ1 investigates the precision and accu-

racy of our proposal, facilitating a comparison among other works. RQ2 compares different task contexts to understand how different combinations can influence SO post reuse.

This paper is structured as follows. Section I presents an introduction of the discussed subject. Section II presents the study and implementation conducted to investigate the association of Stack Overflow posts with project tasks. After, Section III evaluates the study and implementation. Conclusion, discussion, and future work are presented in Section IV.

II. STUDY ON RETRIEVING CURATED STACK OVERFLOW POSTS

Curation is the act of searching and selecting useful SO posts for a given problem during software development. The process of curating SO posts is as follows. First, the developer that has a problem or that needs support creates a search string that may retrieve satisfactory results. Second, the search string is submitted to SO. Third, SO executes the search according to internal algorithms and lists the results. Finally, the developer selects one (or a set of) SO post that can be used as support. Each of these steps can be executed repeatedly until developers are satisfied with the results listed and choose a SO post that meets their needs. Once a solution is chosen, curation is over. If results are not helpful, the developer changes the string, exploring new terms that can indicate useful results.

Frequently, after a solution from a SO post is selected, the post from where the solution was extracted is not associated with the project task and, therefore, cannot be reused by developers dealing with similar or recurring tasks. A growing body of literature recognizes the importance of associating external knowledge with the development [13] [11] [14] [10] [15]. Stack Overflow is the source of support considered in this work, and this support is presented in the form of curated SO posts. Through the identification of similar project tasks, SO posts that were once used to support a project task during its solution could be automatically associated with project tasks with similar context(s), therefore making developers aware of this SO post previously used by another developer. After a preliminary investigation [16], this work was further developed. We researched what comprises the context elements of a project task and implemented a process model using the investigated context in Rapidminer¹.

A. Project Task Context

Project Task Context is a set of elements that compose a project task. Investigating possible project task context elements is essential to contribute to clarifying what information from project tasks are available for similarity comparisons. For completeness purposes, we have taken advantage of both academic and industrial perspectives, including project management tools that support project tasks in software development. It is important to consider other sources in grounding research other than formal literature in software engineering [17], allowing a broader theoretical aspect and bringing practical insights to the work.

¹rapidminer.com

From the academic perspective, after an *ad-hoc* literature review, we found that software engineering is knowledge-intensive due to its dynamism and the massive amount of technology used activity [1] [18]. According to Lindvall and colleagues [18], software engineering has two types of knowledge associated with the project: technical and business domain information. Our work does not consider application domain information; it only considers technical information because it aims to be agnostic to business characteristics. We propose a context element that captures technological information about project tasks (e.g., tags). A tag is a piece of information related to an element. In this case, the elements are any technical information directly related to the task that can characterize the task. Each project has a specific context in respect to product [19], such as technical characteristics that every task will inherit necessarily, indicating the need for a project tag.

From an industrial perspective, we analyzed project management tools, by verifying the default elements each tool has for project tasks. In this *ad-hoc* analysis, we concluded that some of the project task context elements identified in the literature were also reported in software development tools that support project workflows. The analyzed tools were JIRA², Trello³ and Redmine⁴, which are broadly used. After performing an analysis of both literature and project management tools contents, we present a list of the identified context elements in Table I.

TABLE I
CONTEXT ELEMENTS.

Element	Description	Source
Project/Board	The name of the project that tasks belongs to	Redmine, Trello JIRA
Project Tag	Tags related to the project	Lindvall et al. [18]
Process	Process information that can be associated with the task	Lindvall et al. [18]
Title	The title of the task	Redmine, Trello JIRA
Description	The description of the task	Redmine, Trello JIRA
Category	A classification used to divide tasks into different niche	Redmine JIRA
Task Tag	Tags related to the task	Lindvall et al. [18], JIRA, Trello

B. Study Implementation

After project task context elements were investigated in both literature and project management tools, we propose an implementation to obtain the similarities between project task context elements. We implemented a process model that is prepared to receive as input a dataset containing project tasks associated with SO posts, and retrieves similarity indexes from pairs of project tasks and evaluates if the SO posts are the same between project tasks with a high degree of similarity.

²atlassian.com/Jira

³trello.com

⁴redmine.org

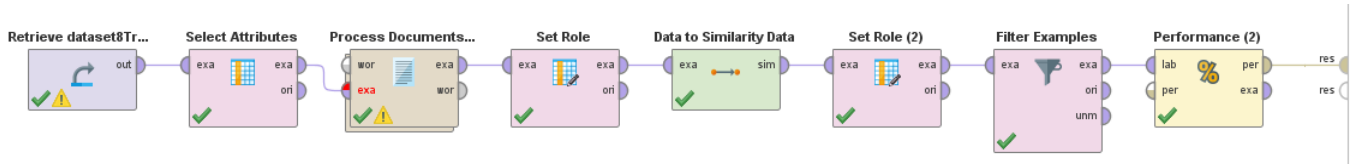


Fig. 1. RapidMiner process model.

RapidMiner (RapidMiner Studio version 8.2) was used to implement a similarity process model. RapidMiner is a compelling data science platform, requiring a small learning curve to be used, widely adopted in the academic field [20]. The implemented RapidMiner model is illustrated in Figure 1. The model loads a dataset of project tasks, pre-processes the text and executes a Jaccard algorithm [21] to retrieve text similarities. More details can be found in the work of [22] and the source code for the implemented model can be found on GitHub (github.com/glauciams/task2stackRapidMiner).

III. EVALUATION

Based on the guidelines proposed by Shani et al. [23] we verify the effectiveness of using task similarities to reuse curated SO posts, through well-established metrics: precision and accuracy. The research questions for the evaluation are:

RQ1: What are the precision and the accuracy metrics for the collected sample? It is essential to verify these metrics to gather quantitative results while ascertaining the effectiveness of considering similar project tasks to reuse curated SO posts. The metrics precision and accuracy were chosen after being the most common metrics identified in the works that relate to ours.

RQ2: What are the impacts in precision and accuracy when different context elements are combined? We evaluate different project task context combinations because task contexts can vary in each project. A project can maintain records of processes and another project might not, for example. Given this variation, it is essential to understand the impacts of different project task context combinations.

Hypothesis: Project task similarity can provide helpful suggestions for curated Stack Overflow posts. We test this hypothesis by verifying whether similar tasks (similarity above 50%) share the same Stack Overflow posts.

Controlling Variables: Considering this study uses only one dataset, having fixed controlled variables is not a concern. We propose a study considering different variable combinations to analyze the effects of the absence or presence of variables on precision and accuracy.

A. Executing the implemented process model

To execute the implemented process model, a dataset with project tasks has to be loaded to the RapidMiner process model. The selected dataset was gathered from a company in Brazil that has been developing software products for more than 20 years and has a total of 30 employees. The software development projects in this company follow agile guidelines, and the project tasks are managed with the support of a project

management tool. We were able to gather 25 project tasks with associated SO posts to each of the 25 tasks. The software developers and managers of the company provided a dataset of project tasks (context elements and SO posts associated with each task).

B. Results, Discussion and Threats to Validity

After executing the evaluation of the dataset, precision and accuracy are calculated, using the Jaccard algorithm. A confusion matrix is generated by the execution of the RapidMiner process model, which supports the extracted results.

Answering RQ1, the accuracy for the given dataset with the elements identified in Section II is 77.78%, and the precision mean 71.60%. Other works with similar characteristics present different precision and accuracy results. Rahman et al. [24] report 11% of precision and 88% of accuracy. The work of Wang et al. [25] reports 62% of precision.

Answering RQ2, Table II presents the context attributes' combination selected, the precision and accuracy extracted for each combination, and changes made in each combination, as it is not easy to perceive from the attribute list which attributes were selected and which were not in the combination.

TABLE II
RQ2 RESULTS: CONTEXT COMBINATIONS

Precision	Accuracy	Combination changes
71.60%	77.78%	Current Work - RQ1
61.17%	70%	Included Interaction
48.01%	54.69%	Removed Process
69.81%	66.67%	Removed Project Tags and Task Tags
45.64%	37.12%	Removed Title and Description
40.87%	36.57%	Removed Title
45.64%	37.12%	Removed Description
54.76%	38.28%	Removed TaskTags
61.46%	74.47%	Removed ProjectTags
49.01%	41.18%	Removed Category
61.46%	74.47%	Removed Project
77.78%	85%	Removed Project and Project Tags
71.67%	54%	Removed Category, Process, Project, Project Tags and Task Tags
13.51%	22.69%	Removed Category, Process, Project, Title and Description

According to the model proposed by Wohlin et al. [26], the **internal threats** to the validity of this evaluation are the small sample size, which can produce distortions in the expected result and conclusions. A strategy to mitigate this problem can include the use of more extensive or various samples. The lack of another sample can also be characterized as an **external threat** because it can jeopardize the generalization of the study results. As **construction threats**, we can cite the

selection of methods and definitions of measures, which was performed according to the related work, but with no other indication of exactitude or representativeness. Finally, the fact that the same developers can be involved in the study might indicate **conclusion threats**, given the possibility the text of project tasks can be standardized, in case they were written by the same person.

IV. CONCLUSIONS

Identifying similar project tasks during software development could decrease the curation effort of developers, when in need of external support. We performed a study to identify project task context elements, propose an implementation of a process model and evaluate the proposal. The implementation aims to verify if similar project tasks are associated with the same SO posts curated by developers. Precision and accuracy were 71.60% and 77.78%, respectively. We also conclude that there is a significant variation of precision and accuracy when project task context elements are added or removed. When considering the project task context elements initially identified in Section II, precision and accuracy are the highest among all combinations. Results also indicate that the hypothesis of the evaluation Project task similarity can provide useful suggestions of curated SO posts is correct, as the prediction and accuracy are as high as 70%. The model built in RapidMiner can be used with other datasets with similar characteristics, indicating the study is replicable.

Future work might involve the development of a recommendation tool using the proposed strategy, allowing the incorporation of rating mechanisms for given suggestions. The tool was not developed in this work, as we intended to focus on the study of the reuse of curated SO posts using task similarity. A deeper understanding of the role of project task context elements should be pursued, allowing the definition of weights for specific contexts.

ACKNOWLEDGMENT

The authors thank the Natural Sciences and Engineering Research Council of Canada (NSERC), the Emerging Leaders in the Americas Program (ELAP) and MITACS.

REFERENCES

- [1] C. Di Ciccio, A. Marrella, and A. Russo, "Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches," *Journal on Data Semantics*, vol. 4, no. 1, pp. 29–57, 2015.
- [2] S. Vasanthapriyan, J. Tian, and J. Xiang, "A survey on knowledge management in software engineering," in *Software Quality, Reliability and Security-Companion (QRS-C), 2015 IEEE International Conference on*. IEEE, 2015, pp. 237–244.
- [3] R. Minelli, A. Mocci, and M. Lanza, "I know what you did last summer - an investigation of how developers spend their time." Piscataway: The Institute of Electrical and Electronics Engineers, Inc. (IEEE), May 1, 2015, p. 25.
- [4] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz, "The work life of developers: Activities, switches and perceived productivity," *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1178–1193, 2017.
- [5] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Leveraging crowd knowledge for software comprehension and development." IEEE, Mar 2013, pp. 57–66.
- [6] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest qa site in the west." ACM, May 7, 2011, pp. 2857–2866.
- [7] S. Fumin, W. Xu, S. Hailong, and L. Xudong, "Recommendflow: Use topic model to automatically recommend stack overflow qa in ide," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 2016, pp. 521–526.
- [8] X. Liu, B. Shen, H. Zhong, and J. Zhu, "ExpSol: Recommending online threads for exception-related bug reports," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2016, pp. 25–32.
- [9] T. P. Sahu, N. K. Nagwani, and S. Verma, "An empirical analysis on reducing open source software development tasks using stack overflow," *Indian Journal of Science and Technology*, vol. 9, no. 21, 2016.
- [10] T. Wang, G. Yin, H. Wang, C. Yang, and P. Zou, "Linking stack overflow to issue tracker for issue resolution." ACM, Nov 17, 2014, pp. 11–14.
- [11] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the ide." IEEE Press, May 18, 2013, pp. 1295–1298.
- [12] M. Rahman and C. K. Roy, "Surfclipse: Context-aware meta-search in the ide," in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 617–620, iD: 1.
- [13] D. Correa and A. Sureka, "Integrating issue tracking systems with community-based question and answering websites," in *2013 22nd Australian Software Engineering Conference*. IEEE, 2013, pp. 88–96.
- [14] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining stackoverflow to turn the ide into a self-confident programming prompter." ACM, May 31, 2014, pp. 102–111.
- [15] P. Kochhar, "Mining testing questions on stack overflow," in *Proceedings of the 5th International Workshop on Software Mining*. ACM, 2016, pp. 32–38.
- [16] G. Melo, U. Telemaco, T. Oliveira, P. Alencar, and D. Cowan, "Towards using task similarity to recommend stack overflow posts," in *Avances en Ingenieria de Software a Nivel Iberoamericano, CIBSE 2018*, 2018, pp. 199–211.
- [17] V. Garousi, M. Felderer, and M. Mäntylä, "The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature," in *Proceedings of the 20th international conference on evaluation and assessment in software engineering*. ACM, 2016, p. 26.
- [18] M. Lindvall and I. Rus, *Knowledge management for software organizations*, ser. Managing software engineering knowledge. Springer, 2003, pp. 73–94.
- [19] F. M. Santoro, P. Brézillon, and R. M. De Araujo, "Context dynamics in software engineering process," in *International Conference on Computer Supported Cooperative Work in Design*. Springer, 2006, pp. 377–388.
- [20] N. Schlitter, J. Lässig, S. Fischer, and I. Mierswa, "Distributed data analytics using rapidminer and boinc," in *Proceedings of the 4th Rapid-Miner Community Meeting and Conference (RCOMM 2013)*, 2013, pp. 81–95.
- [21] P. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*, pearson internat. ed. ed. Boston ; Munich [u.a.]: Pearson Addison Wesley, 2006.
- [22] G. Melo, "Retrieving curated stack overflow posts of similar project tasks," Master's thesis, Universidade Federal do Rio de Janeiro, 2018. [Online]. Available: <https://www.cos.ufrj.br/index.php/pt-BR/publicacoes-pesquisa/details/15/2881>
- [23] G. Shani and A. Gunawardana, *Evaluating Recommendation Systems*, 1st ed., ser. Recommender Systems Handbook. Boston, MA: Springer US, 2011, pp. 257–297.
- [24] M. Rahman, S. Yeasmin, and C. K. Roy, "Towards a context-aware ide-based meta search engine for recommendation about programming errors and exceptions." IEEE, Feb 2014, pp. 194–203.
- [25] T. Wang, G. Yin, H. Wang, C. Yang, and P. Zou, "Automatic knowledge sharing across communities: a case study on android issue tracker and stack overflow," in *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*. IEEE, 2015, pp. 107–116.
- [26] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science Business Media, 2012.

Software Defect Prediction Model Based on Improved Deep Forest and AutoEncoder by Forest

Wenbo Zheng^{†*✉}, Shaocong Mo^{§*✉}, Xin Jin[¶], Yili Qu^{||}, Zefeng Xie^{††} and Jia Shuai^{||}

[†]School of Software Engineering, Xi'an Jiaotong University, Xi'an 710049, China

[§]College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China

[¶]School of Management, Huazhong University of Science and Technology, Wuhan 430074, China

^{||}School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China

^{††}School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China

Abstract—Software defect prediction is an important way to make full use of software test resources and improve software performance. To deal with the problem that of the shallow machine learning based software defect prediction model can not deeply mine the software tool data, we propose software defect prediction model based on improved deep forest and autoencoder by forest. Firstly, the original input features are transformed by the data augmentation method to enhance the ability of feature expression, and the autoencoder by forest performs the data of dimensionality reduction on the features. Then, we use the improved deep forest algorithm and autoencoder by forest to build software defect prediction model. The experimental results show that the proposed algorithm has higher performance than the original deep forest (gcForest) algorithm and other existing start-of-art algorithms, and has higher performance and efficiency than other deep learning algorithms.

Keywords—Software defect prediction, Deep forest, AutoEncoder by forest, Data augmentation.

I. INTRODUCTION

As software systems continue playing a key role in all areas of our society, defects arisen from these software have had a major impact on businesses and the lives of people. However, due to the significant increase in the size and complexity of software code libraries, it has become increasingly difficult to identify defects in software code [1], [2]. The importance and challenges of defect prediction make it an active research area in software engineering [3], [4]. Extensive research has been used to develop predictive models and tools to help software engineers and testers quickly narrow down the most likely defective parts of the software code base [5], [6]. Early defect prediction helps prioritize and optimize the effort and cost of inspections and testing, especially when faced with cost and deadline pressures [7], [8].

Machine learning techniques have been widely used to build defect prediction models [9], [10]. Those techniques derive a number of features (i.e. predictors) from software code and feed them to common classifiers such as Naive Bayes [11], Support Vector Machine [12] and Random Forests [13]. But these methods are all shallow machine learning, and

can not perfectly express the complex relationship between unstructured data. When the amount of data reaches a certain level, the learning ability of shallow algorithm is not as good as the deep learning algorithm.

Due to the huge computational hardware requirements of deep neural networks and the dependence of deep neural networks on a large number of hyper parameters, the software defect prediction methods based on deep neural networks are difficult to train to the optimal degree. Zhi-Hua Zhou [14] put forward a deep forest, make up the blank of the decision tree in the field of deep learning. Deep forests have much less parameters than deep neural network and the advantages of higher classification accuracy. Further, Zhi-Hua Zhou [15] put forward EncoderForest (eForest), a kind of auto-encoder, to do data reduction or feature extraction for training model. Therefore, *why not use deep forest and eForest to build software defect prediction model?*

In this paper, we present software defect prediction model based on improved deep forest and autoencoder by forest. First of all, we propose an improved deep forest algorithm for the lack of multi-grained scanning in deep forests through data augmentation. Then, the improved deep forest algorithm and eForest are applied to software defect prediction problem. Finally, we use this model to experiment with the Eclipse bug dataset [16]. The experimental results show that the proposed algorithm is better than existing start-of-art algorithms and less time than the deep neural network algorithm. The contributions of our paper are as below.

- (1) We apply the deep forest and eForest to software defect prediction problem and get better results.
- (2) In our prediction system, the features are automatically learned through the forests model, thus eliminating the need for manual feature engineering which occupies most of the effort in traditional approaches.
- (3) An extensive evaluation using real open source data provided by Eclipse repository demonstrates the empirical strengths of our model for defect prediction.

The outline of this paper is as follows. *Section II* reviews the works of defect prediction, deep forest and autoencoder by forest. *Section III* describes how our prediction model is

*Wenbo Zheng and Shaocong Mo contribute equally to this study. They are both the corresponding author.

✉E-mail: zwb2017@stu.xjtu.edu.cn, mosc@zju.edu.cn

DOI reference number: 10.18293/SEKE2019-008

built. We report our experiments to evaluate our approach in Section IV. In Section V, we conclude the paper and outline future work.

II. RELATED WORK

A. Defect Prediction

In the field of software engineering, it is impossible to detect and eliminate all software defects by any means of detection and verification. It is impossible to develop a software system without any defects in an actual engineering project. Even if the developer is careful and refined, it cannot be ruled out that there are still some errors or unexpected defects in the software system. Software defect prediction is an important way to rationally use software testing resources and improve software performance. Software defect prediction technology can be used to predict more defects that may also exist as early as possible according to the metrics information and defects found in a software product, then testing and validation resources are allocated based on the result appropriately.

The machine learning based defect prediction technology can comprehensively and automatically learn the model to find defects in the software, which has become the main method of defect prediction. For constructing software defect prediction models, many algorithms such as KNN, neural networks [17], SVM [18], Nave Bayes [19], random forest [20] and ensemble learning method [21] can be used. Moreover, there are mainly three techniques are used for implementing the software defect prediction models, as classification, regression and clustering. However, none of these algorithms can perfectly express the complex relationship between unstructured data. When the amount of data reaches a certain level, the learning ability of shallow structure algorithms is not as good as that of deep structure algorithms.

B. Deep Forest

Zhi-Hua Zhou and Ji Feng [14] propose gcForest (multi-Grained Cascade Forest) shown in Fig. 1(a), a novel decision tree ensemble method. This method generates a deep forest (DF) ensemble, with a cascade structure which enables gcForest to do representation learning. Its representational learning ability can be further enhanced by multi-grained scanning when the inputs are with high dimensionality, potentially enabling gcForest to be contextual or structural aware. The number of cascade levels can be adaptively determined such that the model complexity can be automatically set, enabling gcForest to perform excellently even on small-scale data, which makes it possible to control training costs according to computational resource available. Moreover, contrast to DNNs, the gcForest has much fewer hyper-parameters and its performance is robust in different hyper-parameter settings. From experiments results, gcForest gets excellent performance by using the default setting, competitive to DNNs on a broad range of tasks, even across different data from different domains. Deep forest offers an alternative when deep neural networks are not superior, e.g., when DNNs are inferior to random forest and XGBoost.

C. AutoEncoder by Forest

After gcForest, Zhi-Hua Zhou and Ji Feng [15] propose autoencoder by forest called eforest, which is the first tree ensemble based auto-encoder. As we know, auto-encoding is a significant task; take convolutional neural networks (CNN) as an example, it is achieved by deep neural networks (DNNs) in auto-encoding way. This method presents a procedure for enabling forests to do backward reconstruction by utilizing the Maximal- Compatible Rule (MCR), as show in Fig. 1(b), the rule is defined by the traversing backward decision paths of the trees. Encoding and decoding are the two basic function for an auto-encoder. The eForest makes a tree ensemble to do forward encoding and backward decoding, so it is possible for forest to construct an auto-encoder. In addition, the eForest can be trained in both supervised or unsupervised way.

For encoding tasks, there is no difficulty for forest to do that. Here is the encoding procedure: Once the input data traverse down to the leaf nodes which belong to a trained tree ensemble model contained T trees, the procedure will give back a T dimensional vector.

For decoding task, here is the encoding procedure: Firstly, each path can be identified by the leaf node without uncertainty; Secondly, each path corresponds to a symbolic rule; Thirdly, the Maximal-Compatible Rule (MCR) can be calculated to reconstruct the original sample.

In all, auto-encoding task can be achieved by the eForest's forward encoding and backward decoding operations. Experimental results shows eforest has the following advantages: lower reconstruction error contrast to CNN or MLP based auto-encoder, faster training speed, damage-tolerable and high dataset adaption in same domain.

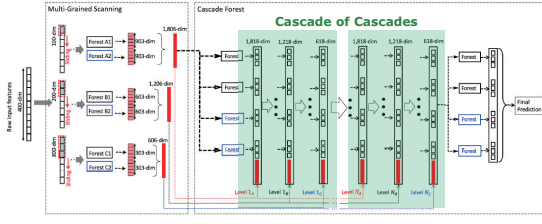
III. SOFTWARE DEFECT PREDICTION MODEL BASED ON FORESTS

We propose our method in this section. Firstly, we propose an improved deep forest algorithm for the lack of multi-grained scanning in deep forests through data augmentation. Secondly, the improved deep forest algorithm and eForest are applied to software defect prediction problem, as shown in Fig. 2.

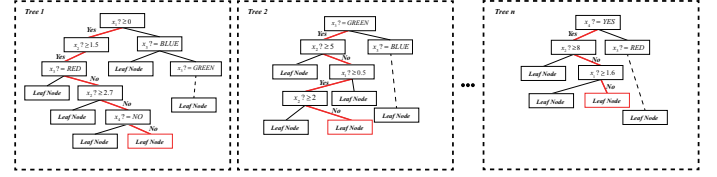
A. Improved Deep Forest Algorithm

Despite the superior performance of deep forests, there are still some shortcomings when using them to build the software defect prediction model.

- Software defect prediction is a binary classification problem, which is to analyze the quality of software modules. According to the classification rules, it is divided into fault-proneness or non-fault-proneness class. Because the main purpose of software defect prediction is to predict whether the modules in the software have fault-proneness modules, so we would set the fault-proneness module as a positive example, and the non-fault-proneness module as a negative example. Multi-grained scanning and cascade structures are used the process that generated class vectors to be aggregated into enhanced feature vectors.



(a) Multi-Grained Cascade Forest [14]



(b) Traversing backward along decision paths [15]

Fig. 1. Deep forest and autoencoder by forest

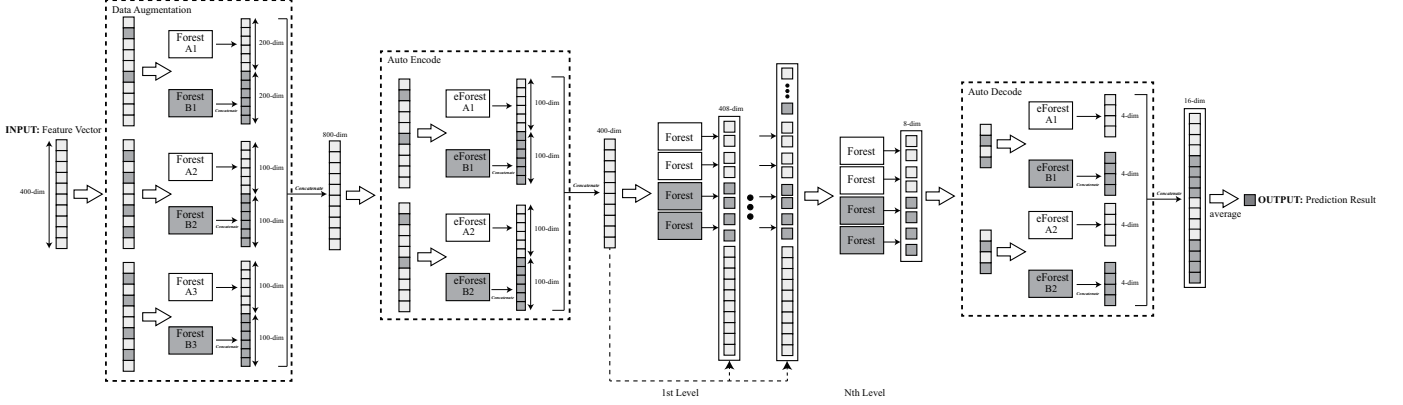


Fig. 2. Software defect prediction model using Forests

But this process can cause redundancy of the feature space. For the software defect prediction problem, the sum of the probability of the module belonging to the positive class and the probability of the module belonging to the negative class is 1, that is, the two probabilities are linear correlation. If two probabilities are both used to fuse the feature vectors, which can cause feature space redundancy and increase the space complexity of the algorithm.

- Multi-grained scanning has significant effects on spatially related features, such as image matching and speech recognition, while features that are spatially uncorrelated (such as software defect prediction, text classification, etc.) may lose important information. The reason is that for spatially uncorrelated features, multi-grained scanning reduces the importance of the the first and the last feature. In the multi-grained scanning process, the first feature and the last feature are only scanning once, that is, these two features are used only once. If the first feature or the last feature is very important, multi-grained scanning can not effectively use this important feature.

To solve the problem that multi-grained scanning in deep forests may lose important information, data augmentation method is used to transform the original input features. Our data augmentation method is to randomly extract features from different original scales from the original input features from the defective/non-defective training samples. The vector by data augmentation is treated as a defective/non-defective instance. The instances which is the same size are combined

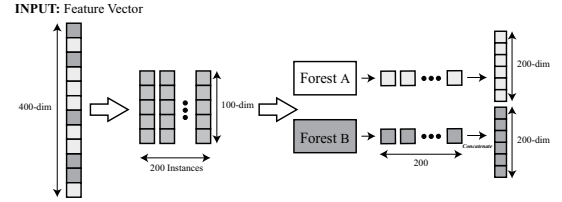


Fig. 3. The process of data augmentation as an example

to form a training entity. The all training entities would train a random forest and a complete random forest, respectively, to predict to generate class vectors. We transform class vectors to enhanced feature vectors.

To be specific, assume that the original input features are 400-dimension, we use randomly sampled methods 200 times, and each sample size is 100-dimensional feature. Each 100-dimensional sample feature is to form an instance. So a total of 200 instances are generated. The 200 instances is called an entity. The entity would be trained using a random The forest and a completely random tree forest, respectively, and then we get two class vectors which is 200-dimension. We transform two 200-dimensional vectors to one 400-dimensional vector, that is, the 400-dimensional original feature vector corresponds to the 400-dimensional enhanced feature vector. The process is shown in Fig. 3.

Similarly, we transform the two 200-dimensional features sampled from each original input feature to generate two 200-dimensional enhanced features vector. We do the concatenate

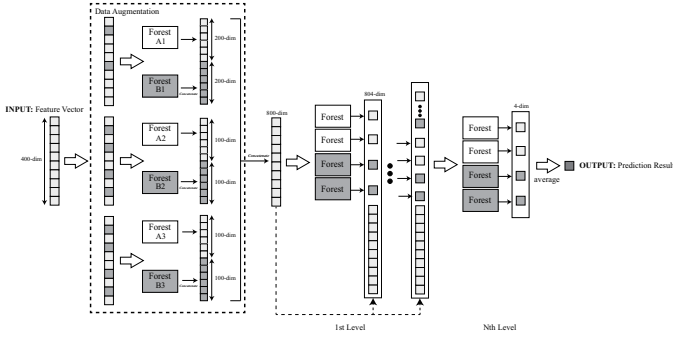


Fig. 4. The pipeline of improved deep forest based on data augmentation

operator to transform the three enhanced feature vectors to a 800-dimensional transformed feature vector. This is the re-representation of the original input features by data augmentation. In other words, each 400-dimensional original feature vector is re-represented by the 800-dimensional transformed feature vector.

The 800-dimensional transformed feature vector would be passed to the cascade forest structure. If each layer of cascade forest consists of 4 forests (two random forests and two completely random forests), the 804-dimensional feature vector would be obtained at the end of the first layer. Then, input the feature vector into the cascade forest structure of the next layer. Repeat the process until the verification performance indicates that the extension of the cascade forest structure should be terminated.

In the test phase, given a test example, we first get the corresponding 800-dimensional transformed feature vector through the data augmentation process, and then predict through the cascade forest structure until the last layer. The final prediction result is decided on the 4 probabilities of 4-dimensional feature vector in the last layer, and the average of the 4 probabilities is the final prediction result. If the average value is greater than or equal to 0.5, the test instance is predicted to be a positive class, otherwise it is a negative class. The process is shown in Fig. 4.

B. Software Defect Prediction Model using Forests

We use data augmentation algorithms to increase the amount of data, challenging the time and efficiency of our algorithm. Therefore, we use the eForest [15] to do data reduction. The process is shown in Fig. 4.

First, we transform original 400-dimensional features to 800-dimensional transformed vectors using data augmentation.

Then, the 800-dimensional transformed feature vector would be passed to the autoencoder using eForest. If autoencoder consists of 4 eForests, the 400-dimensional feature vector would be obtained. The 400-dimensional feature vector would be passed to the cascade forest structure. Because each layer of cascade forest consists of 4 forests (two random forests and two completely random forests) and we reset each forest to output 2-dimensional class vector, the 408-dimensional feature vector would be obtained at the end of

the first layer. Then, input the feature vector into the cascade forest structure of the next layer. Repeat the process until the verification performance indicates that the extension of the cascade forest structure should be terminated. We get a 16-dimensional feature vector through cascade forest structure.

Finally, the 16-dimensional feature vector would be passed to the autodecoder which is the inverse structure of autoencoder. We get 8-dimensional feature vector. The final prediction result is decided on the 8 probabilities of 8-dimensional feature vector in the last layer, and the average of the 8 probabilities is the final prediction result. If the average value is greater than or equal to 0.5, the test instance is predicted to be a positive class, otherwise it is a negative class.

IV. EXPERIMENT RESULTS AND ANALYSIS

Reporting the average of precision/recall across the two classes (defective and clean) is likely to overestimate the true performance, since our dataset is imbalance (i.e. the number of defective files are small). More importantly, predicting defective files is more of interest than predicting clean files. Hence, our evaluation is focus on the defective class.

A confusion matrix is used to store the correct and incorrect decisions made by a prediction model. For example, if a file is classified as defective when it is truly defective, the classification is a true positive (tp). If the file is classified as defective when it is actually clean, then the classification is a false positive (fp). If the file is classified as clean when it is in fact defective, then the classification is a false negative (fn). Finally, if the issue is classified as clean and it is in fact clean, then the classification is true negative (tn). The values stored in the confusion matrix are used to compute the widely-used Precision, Recall, and F-measure.

- Precision: The ratio of correctly predicted defective files over all the files predicted as being defective. It is calculated as:

$$pr = \frac{tp}{tp + fp} \quad (1)$$

- Recall: The ratio of correctly predicted defective files over all of the true defective files. It is calculated as:

$$re = \frac{tp}{tp + fn} \quad (2)$$

- F-measure: Measures the weighted harmonic mean of the precision and recall. It is calculated as:

$$F - measure = \frac{2 \times pr \times re}{pr + re} \quad (3)$$

All experiments were conducted using a 4-core PC with an Intel Core i7 6700HQ with the NVIDIA GTX 1080, 16GB of RAM, and Ubuntu Linux in practice.

To verify the advancement and robustness of our model, we designed two experiments:

- (1) *Ablation Experiment*. In order to verify the advancement and rationality of our model, we experimented with each component.
- (2) *Contrast Experiment*. In order to verify the robustness of our model, in the experiment separately to

TABLE I
THE RESULT OF ABLATION EXPERIMENT ON ECLIPSE BUG DATASET

Method		gcForest [14]				Improved Deep Forest				gcForest [14]+eForest [15]				Ours			
Data Augmentation		×				✓				×				✓			
AutoEncoder/AutoDecoder		×				×				✓				✓			
Training Set	Test set	Accuracy Rate	Precision	Recall	F-measure	Accuracy Rate	Precision	Recall	F-measure	Accuracy Rate	Precision	Recall	F-measure	Accuracy Rate	Precision	Recall	F-measure
file2.0	file2.0	0.8845	0.6708	0.4031	0.5036	0.8847	0.6712	0.4038	0.5042	0.8861	0.6522	0.4601	0.5399	0.8865	0.6753	0.4603	0.5475
	file2.1	0.8555	0.3208	0.2998	0.3099	0.8557	0.3217	0.3001	0.3106	0.8569	0.3361	0.3302	0.3331	0.8572	0.3368	0.3307	0.3337
	file3.0	0.8505	0.4912	0.2832	0.3592	0.8511	0.4912	0.2834	0.3594	0.8427	0.4486	0.273	0.3394	0.8555	0.4492	0.2834	0.3475
file2.1	file2.0	0.8501	0.4543	0.1733	0.2509	0.8505	0.4551	0.1735	0.2512	0.852	0.476	0.2133	0.2946	0.8527	0.4763	0.2137	0.2950
	file2.1	0.8978	0.5705	0.223	0.3206	0.8978	0.5715	0.2238	0.3216	0.8968	0.5446	0.2858	0.3748	0.8988	0.5853	0.2860	0.3843
	file3.0	0.8453	0.4397	0.1652	0.2401	0.8461	0.4407	0.1652	0.2404	0.8453	0.4518	0.2124	0.2889	0.8660	0.4520	0.2125	0.2891
file3.0	file2.0	0.8575	0.5155	0.2728	0.3568	0.8579	0.5158	0.2736	0.3575	0.8575	0.5124	0.3384	0.4077	0.8583	0.5563	0.3390	0.4213
	file2.1	0.8581	0.3221	0.281	0.3002	0.8584	0.3229	0.2811	0.3005	0.8556	0.3277	0.3173	0.3224	0.8656	0.3281	0.3181	0.3230
	file3.0	0.8662	0.5911	0.324	0.4186	0.8667	0.5912	0.3241	0.4186	0.8637	0.5554	0.3974	0.4633	0.8674	0.5958	0.3980	0.4772

realize the Naive-Bayes-Based method [11], Support-Vector-Machine-Based method [12], Random-Forests-Based method [13], Deep Tree-based method [22] and DNN-based method [23], Eclipse standard dataset on the experiment and comparing with the result of the experiment.

Note that in the experiment, the hyper-parameter settings we used were consistent with the reference gcForest [14] and eForest [15].

The data used in this experiment is derived from the Eclipse standard dataset which is one of the most widely used public datasets in software defect prediction research. Since this paper studies the software defect prediction of the classification task, and the Eclipse data set gives the number “post” of defects (refers to the number of defects after the software is released), it is necessary to convert the number “post” of defects into a defective class “hasDefects”. The conversion method is:

$$hasDefect = \begin{cases} 0, & post = 0 \\ 1, & post \neq 0 \end{cases} \quad (4)$$

Since the structure of the “file” level data of the Eclipse data set is same, one of the versions of the data is used as the training data to learn the model, and the three versions of the data can be predicted separately, so that the “file” level data can be used for 9 predictions and verifications.

When the training set and the test set are from the same version, the ten-fold cross-validation is used, that is, the data set is equally divided into 10 parts, one of which is taken as the prediction set, and the 9 remaining data are used as the training set to construct the software defect prediction model and do classification prediction. The experiment was carried out for 10 rounds, and the average of 10 rounds of experiments was taken as the final result.

A. Ablation Experiment

In order to verify the advancement and rationality of our model, we experimented with each component. We use the accuracy rate, precision, recall rate and F-measure to evaluate the experimental results in the experiment. From Table.I, we know our method is better than others. This shows that the design of our algorithm is reasonable.

B. Contrast Experiment

In order to verify the robustness of our model, in the experiment separately to realize the Naive-Bayes-Based method

TABLE II
THE RESULT OF CONTRAST EXPERIMENT ON ECLIPSE BUG DATASET

Method	Training Set	Test set	Accuracy Rate	Precision	Recall	F-measure	Test Time/s
Naive-Bayes-Based	file2.0	file2.0	0.4049	0.2487	0.3818	0.3012	939.10
		file2.1	0.3950	0.1990	0.1361	0.1617	992.42
		file3.0	0.3621	0.1123	0.1366	0.1233	963.81
	file2.1	file2.0	0.7169	0.1784	0.1876	0.1829	965.69
		file2.1	0.5621	0.3514	0.2471	0.2902	981.40
		file3.0	0.4467	0.0499	0.1744	0.0776	905.80
	file3.0	file2.0	0.6412	0.4286	0.1163	0.1830	994.07
		file2.1	0.7742	0.2874	0.3415	0.3121	963.66
		file3.0	0.6147	0.3210	0.1127	0.1668	995.49
Support-Vector-Machine-Based	file2.0	file2.0	0.5266	0.4855	0.3372	0.3980	984.68
		file2.1	0.5802	0.4319	0.1014	0.1642	940.42
		file3.0	0.8175	0.2333	0.1178	0.1565	936.28
	file2.1	file2.0	0.8011	0.1011	0.1754	0.1283	971.73
		file2.1	0.8927	0.2631	0.1463	0.1880	991.80
		file3.0	0.4862	0.1785	0.1936	0.1857	984.54
	file3.0	file2.0	0.6165	0.4705	0.1964	0.2772	910.49
		file2.1	0.7281	0.2550	0.0325	0.0576	996.15
		file3.0	0.6047	0.5585	0.1147	0.1903	944.24
Random-Forests-Based	file2.0	file2.0	0.7175	0.5661	0.3932	0.4641	993.29
		file2.1	0.7424	0.1544	0.2673	0.1957	926.15
		file3.0	0.8227	0.3748	0.2418	0.2940	949.10
	file2.1	file2.0	0.8093	0.4630	0.1572	0.2347	926.51
		file2.1	0.7362	0.3983	0.2712	0.3227	968.88
		file3.0	0.7906	0.2622	0.1532	0.1934	949.81
	file3.0	file2.0	0.6972	0.4031	0.2695	0.3230	986.32
		file2.1	0.6683	0.1613	0.2884	0.2068	988.58
		file3.0	0.6860	0.4845	0.2445	0.3250	954.50
Deep Tree-based	file2.0	file2.0	0.7919	0.2475	0.3052	0.2734	2740.51
		file2.1	0.8236	0.4088	0.1980	0.2667	3462.27
		file3.0	0.8006	0.3925	0.1907	0.2567	2672.78
	file2.1	file2.0	0.8806	0.5172	0.2197	0.3084	2711.65
		file2.1	0.8445	0.4067	0.1393	0.2075	3313.87
		file3.0	0.8333	0.5195	0.2640	0.3501	3251.04
	file3.0	file2.0	0.8070	0.3062	0.2540	0.2777	3168.86
		file2.1	0.8413	0.5205	0.3410	0.4121	2537.39
		file3.0	0.4778	0.4877	0.1443	0.2228	2815.82
DNN-based	file2.0	file2.0	0.8856	0.5209	0.2136	0.3029	4070.51
		file2.1	0.8140	0.1531	0.3253	0.2082	4629.74
		file3.0	0.5592	0.2508	0.0559	0.0915	4449.19
	file2.1	file2.0	0.8438	0.4124	0.0675	0.1161	4643.71
		file2.1	0.8413	0.5248	0.1379	0.2184	4389.90
		file3.0	0.6942	0.4166	0.1111	0.1754	4447.18
	file3.0	file2.0	0.8260	0.5292	0.1614	0.2473	4767.79
		file2.1	0.6791	0.0655	0.1613	0.0932	4486.30
		file3.0	0.8282	0.4512	0.3592	0.4000	4566.45
Ours	file2.0	file2.0	0.8865	0.6753	0.4603	0.5475	2552.15
		file2.1	0.8572	0.3368	0.3307	0.3337	2641.29
		file3.0	0.8555	0.4492	0.2834	0.3475	2974.45
	file2.1	file2.0	0.8527	0.4763	0.2137	0.2950	2666.39
		file2.1	0.8988	0.5853	0.2860	0.3843	2472.26
		file3.0	0.8660	0.4520	0.2125	0.2891	2560.44
	file3.0	file2.0	0.8583	0.5563	0.3390	0.4213	2699.37
		file2.1	0.8656	0.3281	0.3181	0.3230	2868.56
		file3.0	0.8674	0.5958	0.3980	0.4772	2604.83

[11], Support-Vector-Machine-Based method [12], Random-Forests-Based method [13], Deep Tree-based method [22] and DNN-based method [23], Eclipse standard dataset on the experiment and comparing with the result of the experiment. We use the accuracy rate, precision, recall rate, F-measure and the time of test to evaluate the experimental results in the experiment.

From Table. II, we can get two points:

- In terms of performance, our algorithm is optimal in all cases. Therefore, the robustness of our algorithm is optimal among all methods.

- In terms of test time, our algorithm is not the least time in all cases, but our algorithm is the least time in all cases of deep learning. It shows that our algorithm has obvious advantages in efficiency compared to other deep learning algorithms.

From these two points, we know that our algorithm has good robustness and efficiency.

V. CONCLUSION AND FUTURE WORK

In this paper, in order to solve the problem that the shallow machine learning algorithm can not deeply mine the software data features in the current software defect prediction, software defect prediction model based on improved deep forest and autoencoder by forest is proposed. Firstly, the original input features are transformed by the data augmentation method to enhance the ability of feature expression, and the autoencoder by forest performs the data of dimensionality reduction on the features. Then, we use the improved deep forest algorithm and autoencoder by forest to build software defect prediction model. The experimental results show that the proposed algorithm has higher performance than the original deep forest (gc-forest) algorithm, and has higher performance and efficiency than other deep learning algorithms. However, the algorithm does not consider unbalanced data problem. In future, we will be to study how to further improve the prediction performance of the algorithm and consider a solution to the problem of data imbalance in the training set.

ACKNOWLEDGMENT

This paper was supported in part by the National Natural Science Foundation of China (Grant No. 61702386).

REFERENCES

- [1] F. Wu, X. Jing, Y. Sun, J. Sun, L. Huang, F. Cui, and Y. Sun, "Cross-project and within-project semisupervised software defect prediction: A unified approach," *IEEE Transactions on Reliability*, vol. 67, no. 2, pp. 581–597, June 2018.
- [2] S. Huda, S. Alyahya, M. M. Ali, S. Ahmad, J. Abawajy, H. Al-Dossari, and J. Yearwood, "A framework for software defect prediction and metric selection," *IEEE Access*, vol. 6, pp. 2844–2858, 2018.
- [3] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, "Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 6, pp. 534–550, June 2018.
- [4] E. A. Felix and S. P. Lee, "Integrated approach to software defect prediction," *IEEE Access*, vol. 5, pp. 21 524–21 547, 2017.
- [5] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 5, pp. 412–428, May 2018.
- [6] S. Qiu, L. Lu, and S. Jiang, "Multiple-components weights model for cross-project software defect prediction," *IET Software*, vol. 12, no. 4, pp. 345–355, 2018.
- [7] S. Huda, K. Liu, M. Abdelrazek, A. Ibrahim, S. Alyahya, H. Al-Dossari, and S. Ahmad, "An ensemble oversampling model for class imbalance problem in software defect prediction," *IEEE Access*, vol. 6, pp. 24 184–24 195, 2018.
- [8] T. Chen, S. W. Thomas, H. Hemmati, M. Nagappan, and A. E. Hassan, "An empirical study on the effect of testing on code quality using topic models: A case study on software development systems," *IEEE Transactions on Reliability*, vol. 66, no. 3, pp. 806–824, Sept 2017.
- [9] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 811–833, Sept 2018.
- [10] C. Hu, X. Xue, L. Huang, H. Lyu, H. Wang, X. Li, H. Liu, M. Sun, and W. Sun, "Decision-level defect prediction based on double focuses," *Chinese Journal of Electronics*, vol. 26, no. 2, pp. 256–262, 2017.
- [11] T. Wang and W. Li, "Naive bayes software defect prediction model," in *2010 International Conference on Computational Intelligence and Software Engineering*, Dec 2010, pp. 1–4.
- [12] H. Wei, C. Shan, C. Hu, H. Sun, and M. Lei, "Software defect distribution prediction model based on npe-svm," *China Communications*, vol. 15, no. 5, pp. 173–182, May 2018.
- [13] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, "Software defect prediction using feature selection and random forest algorithm," in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, Oct 2017, pp. 252–257.
- [14] Z. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," *CoRR*, vol. abs/1702.08835, 2017. [Online]. Available: <http://arxiv.org/abs/1702.08835>
- [15] J. Feng and Z. Zhou, "Autoencoder by forest," *CoRR*, vol. abs/1709.09018, 2017. [Online]. Available: <http://arxiv.org/abs/1709.09018>
- [16] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, May 2007, pp. 9–9.
- [17] R. Jindal, R. Malhotra, and A. Jain, "Software defect prediction using neural networks," in *Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions)*, 2014 3rd International Conference on. IEEE, 2014, pp. 1–6.
- [18] P. Selvaraj and D. P. Thangaraj, "Support vector machine for software defect prediction," *International Journal of Engineering & Technology Research*, vol. 1, no. 2, pp. 68–76, 2013.
- [19] A. Okutan and O. T. Yildiz, "Software defect prediction using bayesian networks," *Empirical Software Engineering*, vol. 19, no. 1, pp. 154–181, 2014.
- [20] S. G. Jacob *et al.*, "Improved random forest algorithm for software defect prediction through data mining techniques," *International Journal of Computer Applications*, vol. 117, no. 23, 2015.
- [21] T. Wang, Z. Zhang, X. Jing, and L. Zhang, "Multiple kernel ensemble learning for software defect prediction," *Automated Software Engineering*, vol. 23, no. 4, pp. 569–590, 2016.
- [22] H. K. Dam, T. Pham, S. W. Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C. Kim, "A deep tree-based model for software defect prediction," *CoRR*, vol. abs/1802.00921, 2018. [Online]. Available: <http://arxiv.org/abs/1802.00921>
- [23] A. V. Phan, M. L. Nguyen, and L. T. Bui, "Convolutional neural networks over control flow graphs for software defect prediction," in *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, Nov 2017, pp. 45–52.

Multi-project Regression based Approach for Software Defect Number Prediction*

Qiguo Huang ^{★‡}

Chao Ni ^{★‡}

Xiang Chen ^{★☆☆}

Qing Gu [★]

Kaibo Cao [☆]

[★] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

[☆] School of Computer Science and Technology, Nantong University, Nantong 226019, China

Abstract

Software defect prediction can make software quality assurance (SQA) process more efficient, economic and targeted. Previous studies mainly focused on classifying software modules as defect-prone or not. However, predicting the number of defects for a new software module is rarely investigated. Moreover, these studies built models independently for each project, which may ignore the relatedness among multiple projects. To effectively utilize the relatedness, we propose a novel approach MPR (multi-project regression) for SDNP (software defect number prediction). To verify the effectiveness of MPR, we perform experimental studies on 30 real-world projects and compare our approach with 6 state-of-the-art baselines (i.e., LR, NNR, SVR, DTR, BRR and DBR). AAE (Average absolute error) and ARE (average relative error) performance measures are used to evaluate the performance of MPR. The results show MPR can achieve better performance in most cases, which indicates the competitiveness of MPR in the context of SDNP.

1 Introduction

Software defects are introduced unconsciously during the development process of software projects. After the software projects are deployed, defects in the software will produce unexpected behaviors, even cause huge economic loss in worst cases. Therefore project managers want to use software quality assurance methods to detect defects as many as possible. Due to the limitation of testing resources, project managers hope that they can resort to effective methods to identify potential defective modules as early as possible. Software defect prediction [1–3] is one of such effective methods. It helps to identify defective software modules before the testing phase by analyzing some underlying characteristics of the software system and thus subsequently helps in allocating software testing resources optimally and economically. It constructs defect prediction models by mining software repositories (such as version control systems, bug tracking systems) and uses the constructed models to identify potential defective modules in the new projects.

Many studies have been conducted for software defect prediction by using different statistical and machine learn-

ing techniques. Most of these studies have focused on classifying whether a software module has defects or not. That means previous studies discretized defect number of program modules into two categories: defective or non-defective [4]. However, such simple data preprocessing may lead to information loss. In addition, predicting defect number for program modules can assist in sorting program modules and then allocating more testing resources to these modules with more defects. In this way, the allocation of testing resources can be further optimized.

To the best of our knowledge, previous studies mainly focused on classifying software modules as defect-prone or not. Only a few studies built and evaluated models for defect number prediction [5–7]. These studies built models independently for each project, which may ignore the relatedness among multiple projects. In this paper, to fully utilize the relatedness among projects and propose a novel approach MPR (multi-project regression) for defect number prediction. To verify the effectiveness of MPR, we perform experimental studies on 30 real-world open-source projects and compare our approach with 6 state-of-the-art baselines (i.e., LR, NNR, SVR, DTR, BRR and DBR). AAE (Average absolute error) and ARE (average relative error) are used as performance measures. The results show MPR can achieve better performance in most cases.

The main contributions of the paper can be summarized as follows: (1) to our best knowledge, we firstly propose a novel approach MPR for predicting defect number on multiple projects. This approach can utilize the relatedness among projects and builds many predictors simultaneously. (2) we conduct empirical studies on real-world software projects to demonstrate the effectiveness of MPR. The final results show that MPR can achieve a statistical improvement over classical baselines.

2 Related Work

Previous studies mainly investigated regression based methods for this SDNP. Graves et al. [5] considered a generalized linear regression method and conducted empirical studies on a large-scale telecommunication system. They found a significant correlation among module's age, changes made to the modules and the age of changes. Ostrand et al. [6] employed negative binomial regression (NBR) method. Wang and Zhang [8] proposed BugState,

which is based on a defect state transition model. Janes et al. [9] considered three methods (i.e., NBR, zero-inflated NBR, Poisson regression) for 5 real-time telecommunication systems and found that zero-inflated NBR method achieved the best performance. Then Gao and Khoshgof-taar [10] further performed empirical studies on two industrial software projects and their results confirmed the best performance of zero-inflated NBR. Chen et al. [11] conducted empirical studies for 6 regression algorithms. Then found that using decision tree regression can achieve the highest performance in both within-project prediction scenario and cross-project prediction scenario. Yu et al. [12] explored resampling (i.e., SMOTEND and RUSND) and ensemble learning (i.e., AdaBoost.R2) methods. Then they proposed two hybrid methods (i.e., SMOTENDBoost and RUSNDBoost) and these two methods can achieve higher performance. Rathore and Kumar [13] explored the capability of decision tree regression in two different scenarios (i.e., intra-release prediction scenario and inter-release prediction scenario). They [14] compared six methods for SDNP, such as genetic programming, multi-layer perceptron, linear regression, decision tree regression, zero-inflated Poisson regression, and negative binomial regression. Recently, they [15, 16] further considered ensemble learning methods for SDNP.

Based on the above analysis, when performing SDNP for many related projects, we found that the models proposed in previous studies can merely build one model for one project per time and they can not use useful information in other related projects. Different from previous studies, we propose a novel method to build multiple regression models simultaneously for many related projects.

3 Our Proposed MPR Approach

This section first gives the preliminaries on multiple linear regression based on multi-task learning [17]. Then it describes our proposed MPR approach in detail.

3.1 Preliminaries

Definition 1 (*Multiple Liner Regression*) Suppose there are m projects to be learnt $\{P^i\}_{i=1}^m$, where all the projects or at least a subset of them are related. A multiple linear regression model, which aims to improve the learning of a model for P^i by using the knowledge contained in the m projects, can be represented as the following: $Y^i = X^i W^i + B^i, i = 1, \dots, m$, which can build many liner regression models simultaneously.

In this definition, $Y^i = (y_1^i, \dots, y_{n^i}^i) \in \mathbb{R}^{n^i}$ represents the responds for the i -th project, regressed on the training instance matrix $X^i = (x_1^i, \dots, x_{n^i}^i) \in \mathbb{R}^{n^i \times F^i}$, where n^i and F^i represent the number of training instances of the i -th project and the feature space of i -th project respectively. $B^i \in \mathbb{R}^{n^i}$ is a bias vector. There are m tasks in total. For

convenience, we transform these quantities into matrices. That is, $Y \in \mathbb{R}^{n^i \times m}$ represents the responses of the regression model, $W \in \mathbb{R}^{F^i \times m}$ represents the regression parameters and $B \in \mathbb{R}^{n^i \times m}$ represents the noise.

According to this definition, it is not hard to find that the parameter W is very important since it contains the relatedness among different projects. For convenience, we represent the parameters W as a matrix, where each column corresponds to a project, and each row corresponds to a feature. Meanwhile, not only different projects have the common characteristics, but also they have the personalized characteristics. Therefore, both the shared information and non-shared information are included in this parameter W . It seems to be that any one structure might not fully capture all of those information, but a decomposition of the structure might [18]. Therefore the matrix W should be decomposed into two component matrices P and Q , i.e., $W = P + Q$. In particular, some rows of W would have many non-zero entries, which are corresponding to features shared by several projects ("shared information"). We use a row-sparse [18, 19] matrix P to represent such shared information, where each row is either all zero or mostly non-zero, and the number of non-zero rows is small. Some rows of W would be project-sensitive, since they correspond to those features, which are relevant for some projects but not all ("non-shared information"). We use an elementwise sparse matrix Q to represent such non-shared information. It is obvious that some rows of W would have all zero entries, since they correspond to those features that are not relevant to any project.

The proposed method MPR uses the multiple linear model [18] to explicitly model the relatedness among different projects. MPR estimates a sum of the two parameter matrices P and Q with different regularizations for each to encourage row-sparsity in P and elementwise sparsity in Q , therefore MPR can effectively capture shared and non-shared information among related projects.

3.2 The details of MPR Approach

Previous methods can build one regression model for one target project at a time [11, 13, 14], while MPR can build multiple regression models for multiple target projects at a time. Fig. 1 presents the overall framework of MPR. In particular, some data preprocessing operations are performed on these projects. For example, we normalize the numerical features for all these projects. The number of related projects to be learnt equals to the number of models outputted by MPR. Relatedness of different projects is a core concept in multiple linear regression learning. In this paper, we simply consider two type of relatedness. The one is whether these project are developed by the same organization. The other is whether these project have the same feature space.

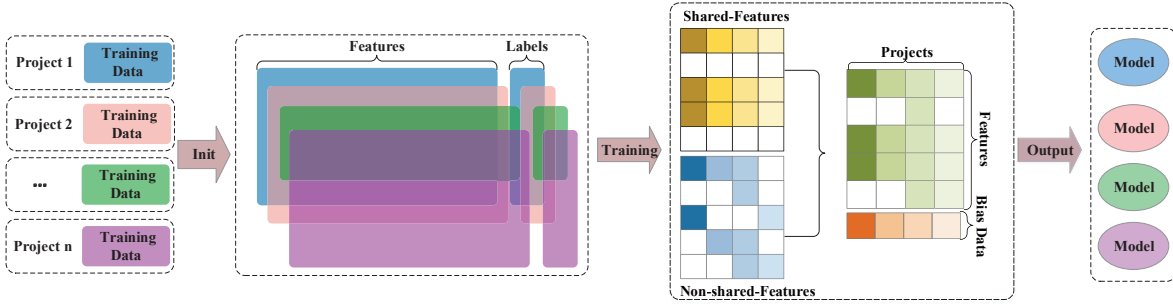


Figure 1: The Framework of MPR

The process of MPR can be described in Algorithm 1. In particular, MPR firstly needs to initialize the start points. In our implementation, we simply initialize the start points with a zero matrix (Line 1). In fact, the starting points can be initialized to a guess value computed from data or a specified point. MPR defines some variables which control the process of searching for optimal value for S , NS and B (Line 2). Then, MPR goes into a loop block with $maxIter$ iteration times (Lines 3-24). In this loop, MPR updates $Shared$, $NonShared$ and $Bias$ with the search factor α (Lines 4-5). Later, it computes performance value and gradients of the current search points (Line 6). Next, MPR goes further into a inner loop in which it can only be terminated when a specific terminating condition is satisfied (Lines 7-18). In this inner loop, the Armijo Goldstein line search scheme [20] is used to compute the optimal shared and non-shared information structure together with the optimal bias data (Lines 8-17). After reaching the termination of the inner loop, it stores previous search points and does preparation for the next iteration. If the difference between the latest two performance meets the requirements of maximum tolerance, MPR can jump out of the outer loop block and return in advance (Lines 19-21). If not, MPR will go to the next iteration with the updated change factor (i.e., t', t) (Lines 22-23). At last, it returns the best value for S , NS and B after given iterations (Line 25).

4 Experiment Design and Result Analysis

4.1 Experimental Subjects

In our empirical studies, 30 projects from PROMISE are used to verify the performances of MPR. These projects can be downloaded from PROMISE and they are widely used in previous empirical studies [11, 12, 15, 16, 21, 22]. The characteristic of these projects are shown in Table 1, which includes project name, number of modules, number (percentage) of defective modules and the maximum defects contained in the modules. Each project use 20 metrics (i.e., features) to measure extracted modules, which are designed

based on the code complexity and the characteristic of object oriented program. Notice that the granularity of extracted program modules is set to class.

Table 1: Statistic of Experimental Subjects

Project (Versions)	# Modules	# Defective Modules	% Defective Modules	Max
ant(1.3,1.4,1.5,1.6,1.7)	125-745	33-338	10.92%-26.21%	(3,3,2,10,10)
camel(1.0,1.2,1.4,1.6)	339-965	14-522	3.83%-35.53%	(2,28,17,28)
ivy(1.1,1.4,2.0)	111-352	18-233	6.64%-56.76%	(36,3,3)
jedit(3.2,4.0,4.1,4.2)	272-367	106-382	13.08%-33.09%	(45,23,17,10)
synapse(1.0,1.1,1.2)	157-256	21-145	10.19%-33.59%	(4,7,9)
xalan(2.4,2.5,2.6)	723-885	156-625	15.21%-48.19%	(7,9,9)
xerces(1.2,1.3)	440-453	115-193	15.23%-16.14%	(4,30)
prop(1.2,3,4,5,6)	660-23014	79-5493	9.6%-15.3%	(37,27,11,22,19,4)

4.2 Performance Measures

To evaluate the performance of MPR for SDNP, average absolute error (AAE) and average relative error (ARE) performance measures are used.

AAE measures the average magnitude of the errors in a set of predictions. It shows the difference between the predicted value and the actual value and is defined as:

$$AAE = \frac{1}{n} \sum_{i=1}^n |\bar{Y}_i - Y_i| \quad (1)$$

ARE calculates how large the absolute error is compared with the total size of the object measured and is defined as:

$$ARE = (1/n) \sum_{i=1}^n |\bar{Y}_i - Y_i| / (Y_i + 1) \quad (2)$$

In Formulation 1 and Formulation 2, \bar{Y}_i is the predicted number of defects for i -th software module, Y_i is the actual number of defects for this module and n represents the number of modules. For ARE, sometimes Y_i may be zero. To mitigate this issue, we add the actual value of Y_i to one at the denominator to make the definition always well-defined [10]. Notice that the smaller the value of AAE and ARE, the better performance of the constructed regression model [23].

4.3 Experimental Setup

To evaluate the performance of MPR, we consider 6 baselines (i.e., LR, NNR, SVR, DTR, BRR and DBR). To

⁰<http://openscience.us/repo/>

Algorithm 1 Multiple Projects Regression (MPR)

Input:

$X^{(n \times F) \times m}$: all the data of related projects;
 $Y^{(n \times 1) \times m}$: the corresponding responses of all related projects;
 $maxIter$: the number of maximum iteration;
 $maxTol$: the tolerance of the performance between two iterations;

Output:

$Shared^{F \times m}(S)$: all shared information among projects;
 $NonShared^{F \times m}(NS)$: non-shared information for each project;
 $Bias^{m \times 1}(B)$: the bias data for regression;

```
1: Initial the start points: filling  $Shared$ ,  $NonShared$ ,  $Bias$  with a zero matrix;  
2: Let  $t=1$ ,  $t'=iter=0$ ,  $\gamma=1$ ,  $\gamma_{inc}=2$ ,  $\alpha$ =Initial search factor, which controls the search for optimal value of  $S$ ,  $NS$ ,  $B$ ;  
3: while ( $iter < maxIter$ ) do  
4:   Set  $\alpha = (t' - 1)/t$ ;  
5:   Find next search point for  $S$ ,  $NS$ ,  $B$  with  $\alpha$ , and store them into  $S_{tmp}$ ,  $NS_{tmp}$ ,  $B_{tmp}$ ;  
6:   Compute the whole loss value and gradients of the current search points with  $S_{tmp}$ ,  $NS_{tmp}$  and  $B_{tmp}$ , and return  $gW_{tmp}$ ,  $gC_{tmp}$  and  $L_{tmp}$ ;  
7:   while (true) do  
8:      $S_t = proximal_{L_{1,\infty}}norm(S_{tmp} - \frac{gW_{tmp}}{\gamma}, \frac{BRP.\alpha}{\gamma})$ ;  
9:      $NS_t = proximal_{L_{1,1}}norm(NS_{tmp} - \frac{gW_{tmp}}{\gamma}, \frac{BRP.\beta}{\gamma})$ ;  
10:     $B_t = B_{tmp} - gC_{tmp}/\gamma$ ;  
11:    Evaluate performance and save it in list of  $funcVal$ ;  
12:    Use the Frobenius norm operation on the delta of  $\{S_t - S_{tmp}, NS_t - NS_{tmp}, B_t - B_{tmp}\}$ , respectively, then sum them up;  
13:    if (reach termination conditions) then  
14:      break;  
15:    else  
16:       $\gamma = \gamma \times \gamma_{inc}$ ;  
17:    end if  
18:  end while  
19:  if ( $abs(funcVal(end) - funcVal(end - 1)) \leq maxTol \times funcVal(end - 1)$ ) then  
20:    break;  
21:  end if  
22:   $iter++$ ;  
23:  Update  $t'$  and  $t$ , i.e.,  $t' = t$ ,  $t = 0.5 \times (1 + \sqrt{1 + 4 \times t^2})$ ;  
24: end while  
25: return  $S$ ,  $NS$ ,  $B$ 
```

avoid the errors in model implementation, we use these regression models implemented by scikit-learn library, which is a popular machine learning toolkit written by python language. Notice our experiments use the default hyper-parameter settings for different regression models. That is, we do not perform hyper-parameter optimization for each regression model. 6 regression models are briefly described as follows:

Linear Regression (LR) is always used to solve the least squares function of the linear relationship between one or multiple independent variables and one dependent variable.

Nearest Neighbors Regression (NNR) is based on the k -nearest neighbors algorithm, and the regression value of an instance is calculated by the weighted average of its nearest neighbors. Then, the weight is set proportional to the inverse of the distance between the instance and its neighbors.

Support Vector Regression (SVR) is the extended from the Support Vector Machines, which only depends on a subset of training data, because the cost function for building a SVR model ignores any training data close to the prediction results of the model.

Decision Tree Regression (DTR) learns simple decision rules to approximate the curve of a given training data set, and then predicts the target variable.

Bayesian Ridge Regression (BRR) is similar to the classical Ridge Regression. The hyper-parameters of such type of models are introduced by prior probability and then estimated by maximizing the marginal log likelihood with probabilistic models.

Gradient Boosting Regression (GBR) is in the form of an ensemble of weak prediction models. Several base estimators are combined with a given learning algorithm to improve the prediction accuracy over a single estimator.

For MPR, we train and test the regression model on all projects simultaneously. Besides, 10-fold cross validation (CV) are used in our experiments. In particular, we split the original dataset into 10 folds equally. Then, the 9 folds are treated as the training data, while the remaining one fold is treated as the test data. For those baselines, we also use 10-fold CV to train and test regression models on all each project independently. To overcome possible bias in the data split process, we perform 10-fold CV for 10 times and report the average performance values.

To check the significance of performance comparison, we conduct Wilcoxon signed-rank test [24], which is a non-parametric statistical hypothesis test. For all the tests, the null hypotheses are that there is no difference between the trained predictors, and the significance level α is set to 0.05. If p -value is smaller than 0.05, we reject the null hypotheses, otherwise we accept the null hypotheses.

4.4 Result Analysis

RQ: How effective is our proposed approach MPR for SDNP? How much improvement can it achieve over the baselines?

Motivation. Our main goal is to propose an approach to improve the performance of model for addressing regression issues by using the knowledge contained in related projects. However, to show the feasibility of this approach, we must analyze how effective it is in its prediction performance and whether it can perform better than baselines. The answer for this RQ would shed light on how much our approach advances the state of the art of SDNP.

Approach. To answer this RQ, we compare MPR with LR, NNR, SVR, DTR, BRR and GBR to investigate whether MPR has competitiveness over these baselines. AAE and ARE are used to evaluate the performance of these approaches. In addition, we run all these approaches 100 times independently.

Results. Table 2 and Table 3 show the comparison results among MPR and 6 state-of-the-art regression approaches (i.e., LR, NNR, SVR, DTR, BRR and GBR) based on AAE and ARE respectively. In both of these two tables, the first column presents the names of projects. Here we assume that all projects have a certain relatedness, since they consider the same features. The remaining 7 columns list the performance values of seven approaches in terms of different performance measures (i.e., AAE and ARE). In each row, the best results are in bold. Notice that all these results of different methods are calculated by 100 independent runs.

Table 2: Comparison of MPR and Baseline Methods on PROMISE Dataset in terms of AAE

Project	MPR	LR	NNR	SVR	DTR	BRR	GBR
ant-1.3	0.32	0.43	0.36	0.44	0.46	0.38	0.45
ant-1.4	0.26	0.40	0.38	0.42	0.42	0.41	0.41
ant-1.5	0.29	0.22	0.19	0.26	0.18	0.21	0.18
ant-1.6	0.28	0.58	0.50	0.68	0.55	0.55	0.51
ant-1.7	0.20	0.49	0.48	0.61	0.58	0.48	0.50
camel-1.0	0.16	0.09	0.07	0.16	0.09	0.08	0.09
camel-1.2	0.15	1.06	1.05	0.96	1.08	1.06	1.01
camel-1.4	0.14	0.59	0.55	0.53	0.63	0.57	0.55
camel-1.6	0.14	0.79	0.79	0.67	0.83	0.77	0.73
ivy-1.1	0.28	1.68	1.65	1.93	1.96	1.39	1.50
ivy-1.4	0.24	0.15	0.12	0.21	0.11	0.13	0.13
ivy-2.0	0.24	0.26	0.23	0.31	0.27	0.23	0.23
jedit-3.2	0.27	1.54	1.56	1.54	1.82	1.46	1.60
jedit-4.0	0.28	0.83	0.88	0.90	1.02	0.81	0.87
jedit-4.1	0.29	0.70	0.77	0.86	0.83	0.69	0.78
jedit-4.2	0.26	0.38	0.41	0.45	0.43	0.39	0.36
synapse-1.0	0.19	0.29	0.21	0.28	0.23	0.24	0.22
synapse-1.1	0.19	0.57	0.53	0.59	0.55	0.53	0.50
synapse-1.2	0.19	0.65	0.60	0.67	0.72	0.59	0.64
xalan-2.4	0.05	0.32	0.30	0.37	0.33	0.31	0.30
xalan-2.5	0.04	0.63	0.59	0.61	0.64	0.64	0.59
xalan-2.6	0.06	0.62	0.56	0.62	0.58	0.64	0.53
xerces-1.2	0.07	0.43	0.38	0.38	0.33	0.43	0.35
xerces-1.3	0.06	0.77	0.54	0.55	0.49	0.74	0.46
prop-1	0.18	0.45	0.39	0.39	0.43	0.45	0.42
prop-2	0.24	0.29	0.25	0.27	0.24	0.29	0.28
prop-3	0.23	0.26	0.25	0.25	0.26	0.26	0.26
prop-4	0.44	0.26	0.23	0.26	0.23	0.25	0.24
prop-5	0.21	0.35	0.32	0.34	0.35	0.35	0.34
prop-6	0.18	0.21	0.19	0.22	0.17	0.20	0.19

Table 3: Comparison of MPR and Baseline Methods on PROMISE Dataset in terms of ARE

Projects	MPR	LR	NNR	SVR	DTR	BRR	GBR
ant-1.3	0.28	0.31	0.23	0.29	0.31	0.26	0.32
ant-1.4	0.24	0.30	0.27	0.31	0.30	0.29	0.31
ant-1.5	0.27	0.18	0.14	0.21	0.13	0.17	0.13
ant-1.6	0.23	0.38	0.30	0.38	0.31	0.36	0.31
ant-1.7	0.15	0.32	0.28	0.34	0.34	0.31	0.30
camel-1.0	0.16	0.07	0.05	0.14	0.06	0.06	0.07
camel-1.2	0.11	0.64	0.60	0.45	0.61	0.64	0.60
camel-1.4	0.11	0.37	0.32	0.28	0.39	0.35	0.34
camel-1.6	0.11	0.49	0.46	0.31	0.47	0.46	0.43
ivy-1.1	0.13	0.74	0.61	0.60	0.70	0.64	0.61
ivy-1.4	0.21	0.10	0.08	0.17	0.08	0.09	0.10
ivy-2.0	0.21	0.19	0.16	0.22	0.20	0.16	0.16
jedit-3.2	0.18	0.97	0.75	0.48	0.83	0.91	0.81
jedit-4.0	0.21	0.52	0.47	0.38	0.58	0.52	0.51
jedit-4.1	0.22	0.42	0.41	0.39	0.42	0.41	0.44
jedit-4.2	0.22	0.26	0.26	0.26	0.27	0.27	0.23
synapse-1.0	0.08	0.23	0.15	0.21	0.17	0.18	0.15
synapse-1.1	0.17	0.39	0.32	0.36	0.37	0.36	0.33
synapse-1.2	0.10	0.41	0.35	0.40	0.42	0.37	0.39
xalan-2.4	0.05	0.22	0.20	0.26	0.22	0.21	0.20
xalan-2.5	0.04	0.42	0.38	0.38	0.39	0.43	0.39
xalan-2.6	0.05	0.40	0.34	0.37	0.34	0.41	0.33
xerces-1.2	0.06	0.30	0.25	0.25	0.22	0.30	0.24
xerces-1.3	0.05	0.52	0.28	0.26	0.25	0.50	0.25
prop-1	0.16	0.29	0.23	0.21	0.25	0.29	0.26
prop-2	0.23	0.20	0.17	0.17	0.15	0.20	0.18
prop-3	0.18	0.18	0.17	0.17	0.17	0.18	0.18
prop-4	0.25	0.18	0.15	0.17	0.14	0.17	0.15
prop-5	0.18	0.24	0.22	0.21	0.24	0.24	0.23
prop-6	0.12	0.15	0.14	0.17	0.13	0.15	0.14

Considering the performance of AAE in Table 2, MPR can achieve better performance in most cases. In particular, MPR achieves 4/5 wins, 3/4 wins, 1/3 wins, 4/4 wins, 3/3 wins, 3/3 wins, 2/2 wins and 4/6 wins on ant, camel, ivy, jedit, synapse, xalan, xerces and prop, respectively. MPR has dominate performance in jedit, synapse, xalan and xerces. However, MPR achieves pool performance (e.g., 0.44 on prop-4). Besides, DTR achieves good performance in three projects (i.e., 0.11 on ivy-1.4, 0.23 on prop-4 and 0.17 on prop-6). Considering the performance of ARE in Table 3, MPR can achieve better performance in most cases. In particular, MPR achieves 3/5 wins, 3/4 wins, 1/3 wins, 4/4 wins, 3/3 wins, 3/3 wins, 2/2 wins and 3/6 wins on ant, camel, ivy, jedit, synapse, xalan, xerces and prop, respectively. MPR also achieves dominate performance in jedit synapse, xalan, xerces and prop. However, MPR obtains pool performance (e.g., 0.27 on ant-1.5). Besides, DTR achieves good performance in four projects (i.e., 0.13 on ant-1.5, 0.08 on ivy-1.4, 0.15 on prop-2 and 0.14 on prop-4).

To check whether the performance differences among MPR and six baseline approaches are significant, we conduct the Wilcoxon signed-rank test. Table 4 shows the results in detail. Considering thirty projects from PROMISE, the p -values of “MPR vs LR”, “MPR vs NNR”, “MPR vs SVR”, “MPR vs DTR”, “MPR vs BRR” and “MPR vs GBR” are all less than 0.05, which indicates that MPR performs statistically better than the baseline methods.

Conclusion: By utilizing the relatedness among projects,

Table 4: p -Value of the Wilcoxon Signed-Rank Test Among MPR and Baseline Methods.

Performance Measure	MPR vs LR	MPR vs NNR	MPR vs SVR	MPR vs DTR	MPR vs BRR	MPR vs GBR
AAE	7.76E-07	1.37E-05	2.08E-07	1.08E-05	3.06E-06	8.19E-06
ARE	5.75E-06	2.65E-04	6.28E-06	9.99E-05	2.48E-05	1.23E-04

MPR can achieve better and stable performance for SDNP in most projects.

5 Conclusion and Future Work

In this paper, we propose a novel approach MPR, which mainly based on multiple linear regression. To show the effectiveness of MPR, we use two widely used regression performance measures (i.e., AAE and ARE) to assess the capability of MPR for SDNP on 30 projects. The results show the competitiveness of MPR when compared with six state-of-the-art baselines.

Acknowledge

Qiguo Huang and Chao Ni have contributed equally for this work. This work was supported in part by National Natural Science Foundation of China (Grant Nos.61373012, 61202006, 91218302, 61321491) and China Scholarship Council (Grant No. 201806190172), which supports overseas studies for Chao Ni to Monash University in Australia.

References

- [1] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.
- [2] C. Ni, W.-S. Liu, X. Chen, Q. Gu, D.-X. Chen, and Q.-G. Huang, "A cluster based feature selection method for cross-project software defect prediction," *Journal of Computer Science and Technology*, vol. 32, no. 6, pp. 1090–1107, 2017.
- [3] C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, "An empirical study on pareto based multi-objective feature selection for software defect prediction," *Journal of Systems and Software*, vol. 152, pp. 215–238, 2019.
- [4] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "The impact of using regression models to build defect classifiers," in *Proceedings of International Conference on Mining Software Repositories*, 2017, pp. 135–145.
- [5] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, 2002.
- [6] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340–355, 2005.
- [7] L. Yu, "Using negative binomial regression analysis to predict software faults: A study of apache ant," *International Journal of Information Technology & Computer Science*, vol. 4, no. 8, pp. 63–70, 2012.
- [8] J. Wang and H. Zhang, "Predicting defect numbers based on defect state transition models," in *Proceedings of Acm-Ieee International Symposium on Empirical Software Engineering and Measurement*, 2012, pp. 191–200.
- [9] A. Janes, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, and G. Succi, "Identification of defect-prone classes in telecommunication software systems using design metrics," *Information Sciences*, vol. 176, no. 24, pp. 3711–3734, 2006.
- [10] K. Gao and T. M. Khoshgoftaar, "A comprehensive empirical study of count models for software fault prediction," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 223–236, 2007.
- [11] M. Chen and Y. Ma, "An empirical study on predicting defect numbers," in *The International Conference on Software Engineering and Knowledge Engineering*, 2015.
- [12] X. Yu, J. Liu, Z. Yang, X. Jia, Q. Ling, and S. Ye, "Learning from imbalanced data for predicting the number of software defects," in *Proceedings of the IEEE International Symposium on Software Reliability Engineering*, 2017, pp. 78–89.
- [13] S. Kumar and S. Kumar, *A Decision Tree Regression based Approach for the Number of Software Faults Prediction*. ACM Sigsoft Software Engineering Notes, 2016, vol. 40, no. 1.
- [14] S. S. Rathore and S. Kumar, "An empirical study of some software fault prediction techniques for the number of faults prediction," *Soft Computing*, vol. 21, no. 24, pp. 7417–7434, 2017.
- [15] —, "Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems," *Knowledge-Based Systems*, vol. 119, pp. 232–256, 2017.
- [16] —, *Towards an ensemble based system for predicting the number of software faults*. Pergamon Press, Inc., 2017, vol. 82, no. 1.
- [17] Y. Zhang and Q. Yang, "A survey on multi-task learning," *CoRR*, vol. abs/1707.08114, 2017.
- [18] A. Jalali, P. D. Ravikumar, S. Sanghavi, and R. Chao, "A dirty model for multi-task learning," in *Proceedings of Neural Information Processing Systems Conference*, 2010, pp. 964–972.
- [19] K. Lounici, M. Pontil, A. B. Tsybakov, and S. V. D. Geer, "Taking advantage of sparsity in multi-task learning," in *Proceeding of The Conference on Learning Theory*, 2009.
- [20] V. Torczon, "On the convergence of pattern search algorithms," *SIAM Journal on optimization*, vol. 7, no. 1, pp. 1–25, 1997.
- [21] M. Yan, Y. Fang, D. Lo, X. Xia, and X. Zhang, "File-level defect prediction: Unsupervised vs. supervised models," in *Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2017, pp. 344–353.
- [22] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information & Software Technology*, vol. 59, no. C, pp. 170–190, 2015.
- [23] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software engineering metrics and models*. Benjamin/Cummings Publishing Company, Inc, 1986.
- [24] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

Cross-Project Defect Prediction via Transferable Deep Learning-Generated and Handcrafted Features

Shaojian Qiu¹, Lu Lu^{1*}, Ziyi Cai¹ and Siyu Jiang²

¹School of Computer Science and Engineering, South China University of Technology, Guangzhou, China

²School of Software Engineering, South China University of Technology, Guangzhou, China

*Corresponding author email: lul@scut.edu.cn

Abstract—Although the machine learning-based software defect prediction (SDP) method has shown promising value in software engineering, yet challenges remain. To improve the performance of SDP, some researchers have used deep learning algorithms to extract the semantic and structural features of the program. However, in more practical cross-project defect prediction (CPDP) tasks, whether deep learning-generated features can be directly used should be explored due to the data distribution shift that usually exists in different projects. In this paper, we propose a Transferable Hybrid Features Learning with Convolutional Neural Network (CNN-THFL) framework to conduct CPDP. Specially, CNN-THFL mines deep learning-generated features from token vectors extracted from programs' abstract syntax trees via convolutional neural network. Furthermore, CNN-THFL learns the transferable joint features simultaneously considering deep learning-generated and handcrafted features by applying a transfer component analysis algorithm. Finally, the features generated by CNN-THFL are fed to the classifier to train a defect prediction model. Extensive experiments verify that CNN-THFL can outperform referential methods on 72 pairs of CPDP tasks formed by 9 open-source projects.

Keywords—Software defect prediction, Cross-project defect prediction, Transfer learning, Semantic feature learning

I. INTRODUCTION

Under the trend of increasing software scale and complexity, how to effectively ensure the reliability of software has become a popular research topic. Software defect prediction (SDP), as a novel proposition of quality assurance technology, has received a great deal of attention from researchers [1]–[3]. It is designed to build a predictive model with the machine-learning model and historical software defect data before detecting the defect-prone modules or files in the software. Effective defect prediction can help software quality assurance teams or code reviewers allocate testing resources more reasonably.

SDP usually consists of two phases: extracting features from program files and training the predictor via the machine-learning [4] method. In previous studies, the discriminant features adopted to construct predictive models were elaborately extracted. These handcrafted features mainly include Halstead features [5] based on operators and operands, McCabe features [6] based on dependencies, and CK features [7] based on the object-oriented concept. In recent years, some researchers [4], [8] have suggested that it is not enough to only consider handcrafted features. To expand the features available in SDP for improving predictive performance, they tried to

use the deep-learning methods (e.g., Deep Belief Network, DBN and Convolutional Neural Network, CNN) to mine the semantic and structural features hidden in the software program. The main idea of these methods is to extract the deep learning-generated features from the token vectors generated by programs' Abstract Syntax Trees (ASTs) and feed to the data with these generated features to the machine-learning classifier to obtain the SDP model. Their experiments show that DBN and CNN methods are superior to the traditional SDP methods that use only handcrafted features in the cross-version defect prediction tasks [9].

In practice, it is often difficult to establish an accurate defect predictor for new projects due to the scarcity of defect labels. To overcome this problem, cross-project defect prediction (CPDP) [7] was proposed as an alternative solution to defect predictors that learn new projects (called target projects) by using labeled data from mature projects (called source projects). Then, in the tasks of CPDP, there is a research question of whether deep learning-generated features extracted from the source project can be directly used for the SDP task of the target project. In [10], Wang et al. explore the effect of DBN-generated features on CPDP tasks; however, their experiments were based only on the assumption that the semantic features generated by DBN can capture the common features of defects, which means the features obtained from a project can be used in other projects. In this paper, we posit that this assumption may not always hold. We suggest that, because the data of the source and target projects usually have different distributions, the deep learning-generated features should not be used directly in CPDP tasks.

Let us use an example to show this distribution discrepancy. As shown in Figure 1, we present the data distribution of two real projects, Apache Lucene v2.4 and Apache Ant v1.7. Using these two projects, we can form a sample based on DBN-generated features (Figure 1a) and a sample based on CNN-generated features (Figure 1b). For a comparable demonstration, we selected two-dimensional features to be drawn on the X and Y axes, respectively. We normalized the data to be displayed with min-max scaling and charted the main distributions in the coordinates. In Figure 1, the discrepancy of the data distribution is clearly shown to exist in the two projects ($Pr(X_s) \neq Pr(X_t)$). As we know, if the distribution of training and test data do not match, we are facing sample selection bias or covariate shift problems that will greatly

affect the performance of the predictive model [11].

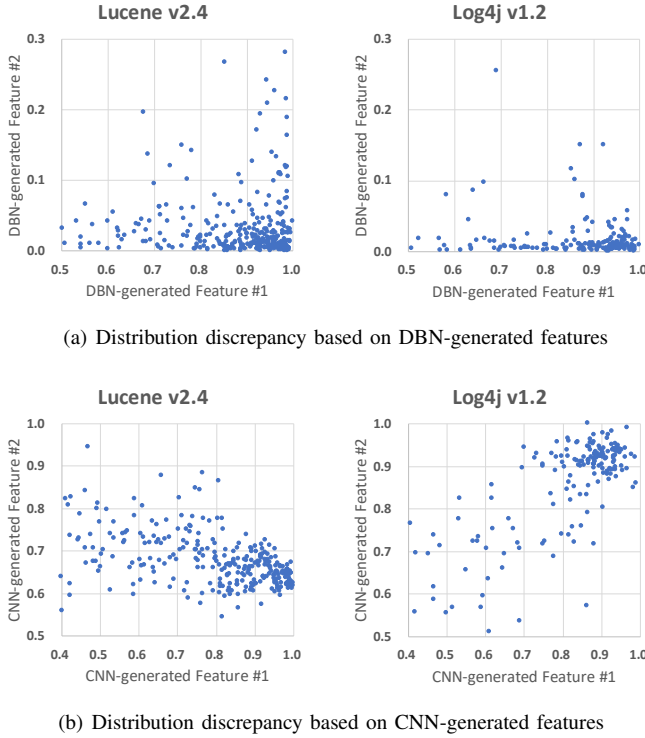


Fig. 1: The distribution discrepancy existed between projects.

To solve this problem, we proposed the Transferable Hybrid Features Learning framework with Convolutional Neural Network (CNN-THFL) to perform CPDP. Specifically, we parsed the source files into ASTs and selected representative nodes on the AST to form token vectors. By converting the token vectors into integer vectors, we adopted CNN to extract the deep learning-generated features. In the process, we concatenated deep learning-generated features with the traditional handcrafted features to construct the hybrid features [4]. To enhance the transferability of hybrid features in CPDP tasks, we attempted to learn transfer components across projects in a reproducing kernel Hilbert space (RKHS) using transfer component analysis (TCA), which could match the data distribution between projects. Finally, the transferable hybrid features generated by CNN-THFL could be fed to the base classifier to train the CPDP model.

The main contributions of this paper are:

- We propose a features-learning framework called CNN-THFL, which jointly considers the transferability of deep learning-generated and handcrafted features in cross-project programs, to handle the distributions discrepancy between projects in CPDP tasks.
- To validate the effectiveness of CNN-THFL, we evaluate it on 72 pairs of CPDP tasks formed by 9 open-source projects. The experimental results show that in terms of average MCC, CNN-THFL improves the TCA by 30.1%

and the DP-CNN by 62.8%.

- By experimental comparison, we found that when using CNN or DBN to mine deep learning-generated features, better results can be attained if this is simultaneously combined with handcrafted features and TCA algorithm, indicating the necessity of using hybrid features and considering their transferability.

We have organized the rest of this paper as follows. In Section II, we review some related works. In Section III, we show the high-level framework of CNN-THFL and elaborate its steps. In Section IV, we provide the experimental setup. In Section V, we show the experimental results to validate the effectiveness of CNN-THFL framework. In Section VI, we discuss the threats to validity. We conclude our work and point out the potential future works in Section VII.

II. RELATED WORK

In this section, we briefly review the related CPDP works.

To effectively apply the SDP technique early in the software lifecycle, CPDP is proposed as a more feasible solution. Its central concept is learning a defect predictor for a new project (target project) by using labeled data from a mature project (source project). In a previous study of CPDP, Zimmermann et al. [12] conducted a large-scale experimental study on its feasibility. They selected 12 real-world projects and analyzed a total of 622 pairs of CPDP tasks. The results show that only 3.4% of tasks were able to achieve acceptable performance. To enhance the performance of the CPDP, Ma et al. [13] developed a transfer Naive Bayes model (TNB) that uses the weighted source data based on target set information to train a weighted Naive Bayes classifier. Nam et al. [14] applied a transfer-learning approach TCA+, which extended TCA [15] with customized normalizing rules, to make data distributions in source and target projects similar. The experimental results show that TCA+ can improve the performance of CPDP by transferable feature learning.

However, the aforementioned methods only use handcrafted features (e.g., Halstead [5], McCabe [6], and CK features [7]). In fact, if we can reasonably mine and use the semantic and structural information of the programs, it will be possible to further improve the performance of SDP. Recently, Wang et al. [10] tried to leverage DBN to automatically learn semantic features using token vectors extracted from the programs' ASTs. Their evaluation of 10 open-source projects shows that the DBN-learned features improve the performance of SDP. Based on the framework of DBN method, Li et al. [4] proposed that CNN is more advanced than DBN in capturing local patterns. They proposed a framework called Defect Prediction via Convolutional Neural Network (DP-CNN) to extract semantic and structural features from token vectors. The experimental results show that on average, DP-CNN improves the performance of SDP. It is worth mentioning that DP-CNN concatenates the CNN-learned feature vectors with traditional handcrafted feature vectors to avoid losing potential

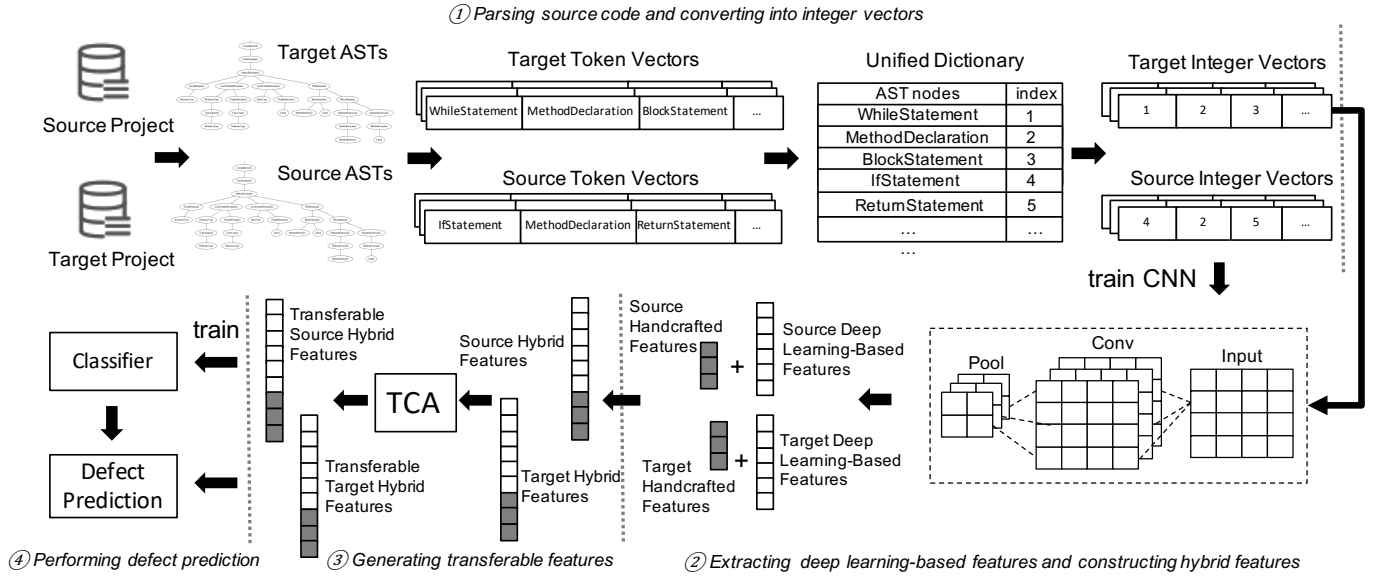


Fig. 2: Overview of our CNN-THFL framework to perform CPDP.

information in the latter. However, neither the DBN nor the CNN has considered the situation that there is distribution discrepancy across projects in CPDP tasks (Wang et al. [10] only assume that semantic features can capture the common features of defects and directly use DBN-generated features in CPDP tasks). Furthermore, this situation is exactly what we want to study and handle in this paper.

III. METHODOLOGY

A. Overall Framework

As presented in Figure 2, our CNN-THFL framework consists of four major steps: 1) parsing source code into tokens and converting them into integer vectors; 2) leveraging the CNN to automatically extract deep learning-generated features and constructing hybrid features; 3) generating transferable hybrid features via TCA; and 4) building classifier and predicting defects.

B. Parsing Source Code and Converting into Integer Vectors

AST is a tree representation of source code, and each structure in the source code can be represented as a node in the tree. Related work [10] has proven that ASTs can be used to detect the integrity and defects of source code. In this paper, we used the *Javalang*¹, an open-source Python package, to parse the Java files and generate corresponding token vectors. Table I lists the node categories and types used in this report. We use the name of the node type as the identifier for each node in the token vector. Considering that the names of methods, classes, and types are project-specific [8], we also used the type name (e.g., MethodDeclarations and ClassInvocation) to label nodes instead of the specific name used in [4].

The token vector extracted by Javalang cannot be directly used as input for CNN model training. To solve this problem,

TABLE I: The selected AST nodes.

Node Category	Node Type
Nodes of method invocations and instance creations	MethodInvocation, SuperMethodInvocation, ClassCreator
Declaration-related nodes	PackageDeclaration, InterfaceDeclaration, ClassDeclaration, ConstructorDeclaration, MethodDeclaration, VariableDeclarator, FormalParameter
Control-flow-related nodes	IfStatement, ForStatement, WhileStatement, DoStatement, AssertStatement, BreakStatement, ContinueStatement, ReturnStatement, ThrowStatement, TryStatement, SynchronizedStatement, SwitchStatement, BlockStatement, CatchClauseParameter, TryResource, CatchClause, SwitchStatementCase, ForControl, EnhancedForControl
Other nodes	BasicType, MemberReference, ReferenceType, SuperMemberReference, StatementExpression,

referring to [4], we established a mapping dictionary between tokens and integers so that the same token would be represented as the same integer. In this way, we could convert the token vectors into integer vectors. Furthermore, because CNN requires input vectors to have the same length, all input vectors would be filled with zero to the length of the longest vector.

C. Extracting Deep Learning-Generated Features and Constructing Hybrid Features

In this study, we applied CNN's feature-generation capability to capture the semantics and local structure of source code [4]. Our CNN included an embedded layer, a convolutional layer, a maximum pool layer, a full-connection layer, and the last output layer as input to the base classifier. All other layers adopted the ReLU activation function except the

¹<https://pypi.org/project/javalang/0.9.2/>

output layer, which used the sigmoid as the activation function. We implemented CNN by *Pytorch*², which has efficient tools for neural networks construction and enables fast, flexible experimentation.

To apply the knowledge carried in the handcrafted features at the same time, we stitched the these features of each project with the deep learning-generated features. The handcrafted features used in this paper were drawn from the metrics used by Jureczko and Madeyski in their defect prediction work [16]. Notice that we used the same handcrafted features from the source and target projects. We spliced deep learning-generated feature vectors with handcrafted feature vectors using the *Concatenate* method of Python to obtain hybrid feature vectors as the input for the next step.

D. Generating Transferable Features

In this study, we hoped to find transferable feature representation to handle the distribution discrepancy between source and target projects. TCA [15] is a transfer-learning method that allows knowledge of defects from a source project to be transferred to a target project. TCA attempts to learn some of the transferable components in the RKHS using maximum mean discrepancy [17]. In the subspace spanned by these transferable components, the properties of source and target data are preserved, and the data distributions in different projects are similar to each other. Therefore, through the new mapping data in this RKHS, we could train the base classifier in the source project, which was also available for the target project.

E. Performing Defect Prediction

For this paper, we chose logistic regression (LR) as a base classifier. We followed the aforementioned steps to process the files in the source and target projects and obtain the transferable hybrid feature for each file. After we fed the TCA-handled data of source project and corresponding label to the LR model, the weights and deviations in our LR would be obtained. Then we used the trained model to predict whether the instances of the target project were defective.

IV. EXPERIMENTAL SETUP

A. Evaluated Projects

To assess the CNN-THFL framework, we collected 9 Java open-source projects from the PROMISE repository, which has been commonly used in recent CPDP researches [18]–[20]. Table II presents the basic information for the 9 projects. Each project consists of a collection of Java files, their corresponding 20 static code attributes (the detailed descriptions of which can be found in [18]), and a label (defective or clean). To verify the generality of our approach, the data sets were composed of several projects with different sizes (ranging from 205 to 815) and defective rates (a minimum value of 11.4% and a maximum value of 98.8%). In our CPDP task, given one of the projects as the training data, another eight projects could

TABLE II: The 9 projects selected from the PROMISE repository.

Project Name	Project Version	Instance Count	Defect Rate
Ant	1.7	745	22.3%
Camel	1.6	965	19.5%
Ivy	2.0	352	11.4%
Log4j	1.2	205	92.2 %
Lucene	2.4	340	59.7%
Synapse	1.2	256	33.6%
Velocity	1.6.1	229	34.1%
Xalan	2.7	909	98.8%
Xerces	1.4.4	588	74.3%

be used as the test data, respectively (e.g., using the data of Ant v1.7 as a training set and the data of Camel v1.6 as the test set). Thus, 72 pairs of CPDP tasks could be performed in this study.

Software defect data have the typical characteristic of imbalanced distribution [21], [22], with the number of minority instances being less than the number of majority instances (e.g., in the Ivy-v2.0 project, the number of defective instances is far less than that of clean instances). In this study, we used the method of random oversampling to avoid imbalanced data that would degrade the performance of our model.

B. Evaluation Metrics

The metric of predictive performance is very important. Although the F1-score has been widely used in recent years [18], [19], we believe that it has problems in SDP tasks [3], especially when there are imbalanced data sets. For example, F1-score excludes true negatives (TN) from calculations, which may be problematic. The test manager will be happy to know whether the component is truly defect-free. Thus, we adopted the Matthews Correlation Coefficient [23] (MCC), as a metric of predictive performance.

As shown in Table III, there may be four outputs when using the dichotomous classifier in the SDP task.

TABLE III: Confusion Matrix

	Truly Defective	Truly Clean
Predictively Defective	TP	FP
Predictively Clean	FN	TN

MCC is the geometric mean of the regression coefficients of the problem and its dual. Based on the confusion matrix, MCC is calculated by the following formula:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

As a correlation coefficient, MCC measures the relationship between predictive class and true class by taking into account all components of confusion matrix. Its return value is on a scale [-1,1] where 1 means a perfect positive correlation and

²<https://pytorch.org>

-1 means a perfect negative correlation. Specifically, MCC can take into account TN, and it is less sensitive to the imbalanced data set.

C. Comparative Methods

We compared CNN-THFL with 9 methods including:

- **LR**. Traditional method, which builds a LR classifier only using handcrafted features.
- **TCA**. A classic transferable features learning method [15].
- **DBN**. A standard DBN model to extract semantic features for SDP [10].
- **DBN-TCA**. A variant of DBN, which applies TCA to obtain transferable DBN-learned features.
- **DBN-DP**. An improved version of DBN proposed by [4], which concatenates the DBN-learned features with the handcrafted features.
- **DBN-THFL**. A variant of CNN-THFL framework which adopted DBN to generate deep learning features.
- **CNN**. A SDP method that extracts deep learning-generated features via standard CNN.
- **CNN-TCA**. A variant of CNN that applies TCA to obtain transferable CNN-learned features.
- **CNN-DP**. A state-of-the-art SDP method that is an improved version of CNN proposed by [4].

Regarding the implementation of DBN, we adopted the same network architectures and parameters as in [10], i.e., 10 hidden layers and 100 nodes in each hidden layer. When implementing CNN, referring to [4], we set the batch size as 32, the epoch number as 15, the embedding dimension as 30, the number of hidden nodes as 100, the number of filters as 10, and filter length as 5. To make fair comparisons, we followed the same code-parsing process to generate integer vectors for neural networks. For TCA, we used the source code provided by its author [15]. For LR, we used the same implementation of *LogisticRegression* in *sklearn.linear_model*, and we adopted default parameters settings by *sklearn*. Considering the process of random oversampling and batch shuffle involve randomness, we conducted each method 20 times, recording their average result of MCC.

V. RESULTS

Because some data sets tend to produce over- or under-performing models, we adopted the Scott-Knott ESD [24], [25] test to compare the performance of the methods we examined (see Figure 3). The Scott-Knott ESD test used here is a mean comparison method that leverages hierarchical clustering to divide a set of MCCs into statistically distinct groups with non-negligible differences. The approach of this test consists of two steps: (1) finding the partitions that maximize the MCC means between groups, and (2) splitting into two groups or merging them together in one group. A detailed description of the Scott-Knott ESD test can be seen in [24]. We can use the *sk_esd* function of the *ScottKnottESD*³ R package to implement the test easily and quickly.

³<https://github.com/klainfo/ScottKnottESD>

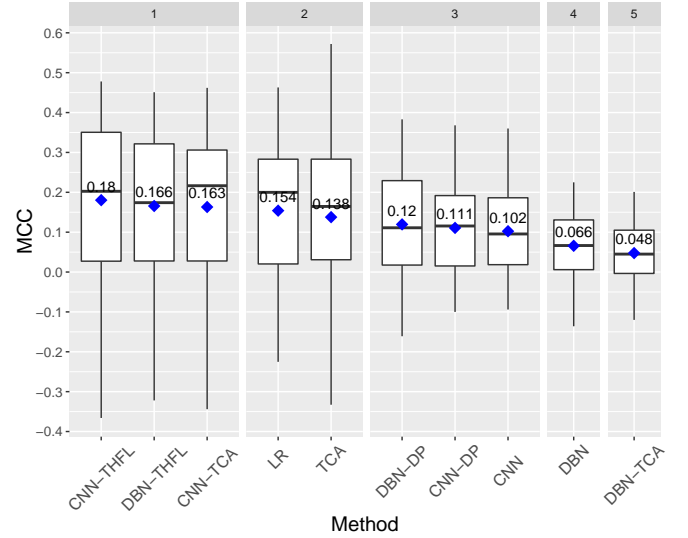


Fig. 3: The Scott-Knott ESD ranking of 10 different methods across the 72 CPDP tasks. The blue diamond indicates the average MCC of our studied methods.

Figure 3 presents the Scott-Knott ESD test of the MCC results (including for 72 CPDP tasks). By comparing 9 referential methods, the average MCC of CNN-THFL was 0.18, which respectively outperformed DBN-THFL, CNN-TCA, LR, TCA, DBN-DP, CNN-DP, and CNN by 8.9%, 10.5%, 17.1%, 30.1%, 50.8%, 62.8%, and 76.1%.

The following two points can be observed from Figure 3:

- 1) DBN-DP and CNN-DP, which consider concatenating the deep learning-generated features with the handcrafted features, will perform better than pure DBN and CNN.
- 2) Better prediction performance (in terms of MCC) can be obtained by combining the THFL framework with deep-learning methods. Among them, CNN-THFL can perform better.

In summary, our CNN-THFL framework improves the performance of CPDP tasks with the consideration of distribution discrepancy between projects. It is worth mentioning that not only CNN-THFL but also DBN-THFL will achieve better performance than the methods without THFL, so we recommend using hybrid features and adapting the distribution discrepancy between projects when performing CPDP. We believe that this improvement of SDP would provide more effective help to test teams for detecting software defects and reasonably allocating test resources.

VI. THREATS TO VALIDITY

A. Implementation of DBN and DP-CNN

In this study, we compared the DBN method [10] and DP-CNN method [4], which are the state-of-the-art deep learning-based SDP methods. Because their implementations have not been publicly released, we tried to reimplement the corresponding methods by *Pytorch* with the same network

structures and parameters. However, we still cannot guarantee that the effects of DBN and DP-CNN that we re-implemented are exactly the same as those in [10] and [4]. However, in our experiments, we used the unified processes (e.g., code-parsing and oversampling steps) and tools (e.g., *Pytorch* and LR classifier) to implement the CPDP framework. As such, our comparative experiment should be fair.

B. Experimental Results Might Not Be Generalizable

In our experiments, 9 open-source projects were selected from the PROMISE repository to access the CPDP methods. The experimental results of these 9 projects (72 CPDP tasks) do not represent all cases. For some other programming languages and commercial software, our proposed method might obtain better or worse results.

C. MCC Might Not Be the Only Appropriate Measure

In our work, we used MCC as the metric of predictive performance. In fact, other metrics (e.g., F-score and G-measure) can also be used in the SDP task. In this paper, we recommend using MCC because it can take TN into account, and it is less sensitive to the problem of imbalanced data sets.

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed a features learning framework called CNN-THFL to improve the performance of CPDP. CNN-THFL aims to mine both transferable deep learning-generated and handcrafted features between source and target projects. An important advantage of CNN-THFL is that it explores the necessity for distribution adaptation of hybrid features that concatenate deep learning-generated features with the traditional handcrafted features in a CPDP task. CNN-THFL is robust to differences in distribution between projects. A large number of experiments on 9 projects with 72 CPDP tasks have been carried out to verify that the proposed framework can achieve better performance in terms of MCC than the advanced referential methods. In future work, we plan to investigate the up-to-date distribution adaptation method to reduce the distribution discrepancy between projects. In addition, we will try to solve the defect prediction across multiple projects with CNN-THFL.

ACKNOWLEDGMENT

This research was supported by the National Nature Science Foundation of China (No. 61370103), Guangzhou Produce & Research Fund (201802020006) and Zhongshan Produce & Research Fund(2017A1014).

REFERENCES

- [1] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, vol. 58, pp. 388–402, 2015.
- [2] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, pp. 170–190, 2015.
- [3] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering*, 2018.
- [4] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 318–328, IEEE, 2017.
- [5] H. H. Maurice, "Elements of software science (operating and programming systems series)," 1977.
- [6] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.
- [7] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.
- [8] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 297–308, IEEE, 2016.
- [9] Z. Xu, J. Liu, X. Luo, and T. Zhang, "Cross-version defect prediction via hybrid active learning with kernel principal component analysis," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 209–220, IEEE, 2018.
- [10] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, 2018.
- [11] J. Huang, A. Gretton, K. Borgwardt, B. Schölkopf, and A. J. Smola, "Correcting sample selection bias by unlabeled data," in *Advances in neural information processing systems*, pp. 601–608, 2007.
- [12] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *The 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 91–100, ACM, 2009.
- [13] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.
- [14] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 382–391, IEEE, 2013.
- [15] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2011.
- [16] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, p. 9, ACM, 2010.
- [17] K. M. Borgwardt, A. Gretton, M. J. Rasch, H. P. Kriegel, B. Schölkopf, and A. J. Smola, "Integrating structured biological data by kernel maximum mean discrepancy," *Bioinformatics*, vol. 22, no. 14, pp. e49–e57, 2006.
- [18] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *IEEE Transactions on software Engineering*, vol. 42, no. 10, pp. 977–998, 2016.
- [19] X. Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 321–339, 2017.
- [20] S. Qiu, L. Lu, and S. Jiang, "Multiple-components weights model for cross-project software defect prediction," *IET Software*, vol. 12, no. 4, pp. 345–355, 2018.
- [21] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Negative samples reduction in cross-company software defects prediction," *Information and Software Technology*, vol. 62, pp. 67–77, 2015.
- [22] S. Qiu, L. Lu, S. Jiang, and Y. Guo, "An investigation of imbalanced ensemble learning methods for cross-project defect prediction," *International Journal of Pattern Recognition and Artificial Intelligence*, 2019.
- [23] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen, "Assessing the accuracy of prediction algorithms for classification: an overview," *Bioinformatics*, vol. 16, no. 5, pp. 412–424, 2000.
- [24] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2017.
- [25] S. Herbold, "Comments on scottknottesd in response to" an empirical comparison of model validation techniques for defect prediction models"," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1091–1094, 2017.

An Investigation of Ensemble Approaches to Cross-Version Defect Prediction

1st Xiaoxing Yang, 2nd Xin Li, 3rd Wushao Wen

School of Data and Computer Science

Sun Yat-Sen University

Guangzhou, China

(yangxx27@mail,lixin49@mail2,wenwsh@mail).sysu.edu.cn

4th Jianmin Su

School of Physics and Optoelectronic Engineering

Guangdong University of Technology

Guangzhou, China

jackysura@163.com

Abstract—Software defect prediction can help software testers to focus on software modules with more defects. Many ensemble methods have been proposed for software defect prediction to divide software modules into defect-prone and defect-free, and these ensemble methods have been proved to be more effective than single learning algorithms. A few ensemble approaches have been applied to predict the number of defects in software modules, and they also perform well in most cases. The good performance of ensemble approaches implies that ensemble algorithms might not only improve the accuracy of software defect classification models, but also improve the performance of defect ranking models. Therefore, we propose an ensemble method based on Yang et al.'s learning-to-rank approach in this paper. Experimental results show that the learning-to-rank-based ensemble approach performs better than the single learning-to-rank approach, which means that the idea of ensemble can improve the performance of the learning-to-rank approach to sort modules in order of defect count. We also conduct a comparison study of ensemble approaches for cross-version defect prediction over 30 sets of cross-version data, which indicates that the ensemble technique of random subspace is more appropriate than boosting over these experimental data sets.

Index Terms—Software defect prediction; Ensemble approaches; Ranking task; Learning-to-rank

I. INTRODUCTION

Software testing activities play a key role in software development, which consume a great amount of resources including time, money and personnel [1]. Timely detecting and repairing defects before releasing the products are critical for software quality assurance [2], [3]. Software defect prediction (SDP) employs software metrics (also referred to as features or attributes) to predict the defect information of software modules (classes, files, etc) in order to support software testing activities [4]. The most frequently investigated tasks of SDP include the classification task [3] and the ranking task [4]. We focus on the ranking task in this paper.

SDP for the ranking task, by sorting software modules in order of defect count, can help testers to focus on software modules with more defects and identify defects more quickly [1], [5]–[7]. Firstly, data is collected from software modules according to software metrics, such as 'response for a class' [8]. Subsequently, modelling approaches are used to construct

a model based on data from modules with known defect numbers. Finally, the model is used to predict defects of software modules with unknown defect numbers, and thus an order of these modules based on the predicted values is obtained, which can help allocate testing resources (for instance, more testing resources for modules with more defects) [7].

Many methods have been employed to construct SDP models for the ranking task, such as linear regression [9], negative binomial regression [1], recursive partitioning, Bayesian additive regression trees, and random forest [10]. Yang et al. proposed an effective learning-to-rank (LTR) method for this task [4], which constructs SDP models by directly optimizing the ranking performance. That random forest [4], [10] performed well for this task implies the usefulness of ensemble algorithms to improve the performance of defect ranking models. Therefore, in this paper, we investigate an ensemble method based on Yang et al.'s LTR approach, in order to see whether the idea of ensemble can benefit this approach.

Considering that the use of ensemble methods for SDP with ranking task was explored by very few researchers, Rathore and Kumar presented a study of different ensemble methods for the prediction of number of defects over three Eclipse data sets [11]. To further compare the ensemble approaches for this task, we conduct a comparison study of ensemble approaches over 30 sets of cross-version data [7].

The main contributions of this paper include: (a) a novel ensemble method based on Yang et al.'s LTR approach, and a comparison study of the proposed ensemble approach with the single LTR approach [4]; and (b) a comparison study of ensemble approaches for cross-version defect prediction for the ranking task.

The rest of this paper is organized as follows. Section 2 presents related work. In Section 3, we describe the proposed ensemble method. The experimental methodologies are detailed in Section 4 and experimental results are reported in Section 5. Section 6 presents the threats to validity, and Section 7 draws the conclusions.

II. RELATED WORK

Data and model construction methods are key factors for software defect prediction (SDP). Because we focus on en-

Corresponding author: Xiaoxing Yang (yangxx27@mail.sysu.edu.cn). DOI reference number: 10.18293/SEKE2019-113

semble approaches for cross-version defect prediction for the ranking task, this section mainly presents the related work from two aspects: methods for constructing SDP models for the ranking task, and ensemble methods in SDP.

A. Methods for Constructing SDP Models for the Ranking Task

Compared with methods for constructing defect prediction models for the classification task, there are fewer approaches particularly proposed for constructing defect prediction models for the ranking task. Most studies employed existing methods to construct models to predict the number of defects. For example, Yang and Wen [7] used two penalized regression methods to construct prediction methods, and experimental results showed that both penalized regression performed better than linear regression and negative binomial regression for cross-version defect prediction. Gao et al. [12] compared eight count models, and the comparative study showed that zero-inflated negative binomial regression, hurdle negative binomial regression and hurdle Poisson regression with threshold 2 were more effective. Weyuker et al. [10] compared four approaches (negative binomial regression, random forest, recursive partitioning and Bayesian additive regression trees) for predicting the number of defects in software modules, and the study showed better performance of negative binomial regression and random forest models than the other two models. Considering that existing SDP models, which are optimized to predict explicitly the number of defects in a software module, might fail to give an accurate order because of the difficulty to predict the exact number of defects in a software module due to noisy data, Yang et al. [4] proposed a LTR approach to directly optimize the ranking performance measure, and the experimental results showed that the LTR approach and random forest were better than other methods to construct SDP models for the ranking task.

B. Ensemble Methods in SDP

Most existing work of ensemble methods for SDP dealt with classification problems instead of ranking problems. For example, Wang and Yao [13] investigated different types of class imbalance learning methods, including re-sampling techniques, threshold moving, and ensemble algorithms, and they found that AdaBoost.NC showed the best overall performance in terms of the measures including balance, G-mean, and area under the curve. The authors also proposed a dynamic version of AdaBoost.NC to adjust parameter automatically during training. Kumar et al. [14] applied ensemble methods to develop models to predict whether software modules are defect-prone or not using 45 datasets from the PROMISE repository. They concluded that ensemble method learning algorithm outperformed individual classifiers. Yohannese et al. [15] evaluated the capability of ensemble learning algorithms in SDP for the classification task using eight NASA software defect data sets. The experimental results revealed that the combined technique could improve the performance. There

are other studies of ensemble methods for SDP for the classification task [16]–[18].

Compared with the work of ensemble methods for defect prediction for the classification task, there exist fewer studies of ensemble methods for defect prediction for the ranking task. A commonly used ensemble method in SDP for the ranking task is random forest. Random forest has been proved to be effective in many studies [4], [10]. Yu et al. [19] explored the potential of using re-sampling techniques and ensemble learning techniques to learn from imbalanced defect data for predicting the number of defects. They studied the use of an ensemble learning technique (i.e., the AdaBoost.R2 algorithm) to handle imbalanced defect data for predicting the number of defects. Experimental results showed the effectiveness of these approaches. Rathore and Kumar [11] performed an empirical study of different homogeneous ensemble methods for predicting the number of faults based on three Eclipse datasets. Experimental results showed that overall ensemble methods produced better performance than a single fault prediction technique. According to their conclusion, random subspace produced better results than other ensemble methods such as boosting and bagging, and decision tree regression was a better base learning technique than multilayer perceptron and linear regression. These studies imply that ensemble algorithms might not only improve the accuracy of software defect classification models, but also improve the performance of defect prediction models for the ranking task. Therefore, we want to investigate the ensemble method that uses the LTR approach as the base learning technique, to investigate whether the idea of ensemble can improve the LTR approach for SDP.

III. THE LEARNING-TO-RANK-BASED ENSEMBLE APPROACH

In this section, we briefly describe Yang et al.'s learning-to-rank (LTR) approach [4]. Subsequently, we present our proposed ensemble method based on this approach.

A. Yang et al.'s Learning-to-Rank Approach

Given a vector of metrics of a software module $\mathbf{x} = (x_1, x_2, \dots, x_d)$, (x_i : the i^{th} metric, and d : number of metrics), the goal of SDP for the ranking task is to predict its relative defect number, which is denoted as $f(\mathbf{x})$ as Equ.(1).

$$f(\mathbf{x}) = \sum_{i=1}^d \alpha_i x_i \quad (1)$$

where α_i s are parameters obtained by training. Once α_i s are fixed, the model is learned. The LTR approach optimizes performance of defect prediction models directly to obtain the parameters using composite differential evolution (CoDE) [20]. Details can be found in Yang et al.'s study [4].

B. The Learning-to-Rank-Based Ensemble Approach

When adopting the LTR model as the base learner, the values of $f(\mathbf{x})$ can vary greatly in the same ranking performance. For this reason, when computing the final output combining all base learning models, we use sigmoid function

to transform the predicted values by the base learning models. When the values of $f_k(\mathbf{x})$ (the predicted values achieved by the k^{th} base learning models) are large, the values of the corresponding sigmoid function are similar. In order to better distinguish these values, we set a coefficient $C1$ for $f_k(\mathbf{x})$, which will not affect the monotonicity. In addition, we set different weights for different base learning models according to their performance over the training set. Assuming that P_k is the training performance of model $f_k(\mathbf{x})$, the final output $G1(\mathbf{x})$ that combines all base learning models is computed as Equ.(2):

$$G1(\mathbf{x}) = \sum_{k=1}^{N1} P_k^W * S(f_k(C1 * \mathbf{x})) \quad (2)$$

$$= \sum_{k=1}^{N1} P_k^W / (1 + \exp^{-C1 * f_k(\mathbf{x})})$$

where $N1$ is the number of base learning models, and W is the power that controls the contributions of different base learning models to the final output according to their training performance. When W equals to 0, the contributions of all base learning models to the final output are the same, regardless of the training performance of these base learning models.

Rathore and Kumar's study [11] showed that random subspace produced better results than other ensemble methods such as boosting and bagging, and random forest [21] (which adopts random subspace) has been proved to be effective in many studies [4], [10]. Therefore, we learn from the selection process of random forest. Assuming that there are d metrics, a subset of $N2$ metrics are selected randomly. Subsequently, the best $N3$ metrics are selected from the $N2$ metrics for constructing base learning models. In this paper, we adopt fault-percentile-average (FPA) [10] as the model performance measure, which is detailed in Section IV. The effectiveness of one metric can be evaluated by directly computing the FPA value of the model based on the metric.

The detailed process of the learning-to-rank-based ensemble approach (LTRE) are shown as follows.

1) **Input:**

M training vectors: $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})$, i : 1 to M , d : number of metrics; Number of base learning models: $N1$; Number of alternative metrics: $N2$; Number of used metrics: $N3$; Coefficient: $C1$; Power: W .

2) **Process:**

- a) for $j = 1, 2, \dots, d$
 - i) compute the performance of model based on training data with only the j^{th} metric, and obtain the corresponding performance F_j .
- end for
- b) for $k = 1, 2, \dots, N1$
 - i) select $N2$ metrics from all metrics randomly
 - ii) select the best $N3$ metrics from the selected $N2$ metrics according to F_j values

- iii) use the training vectors with the best $N3$ metrics to train a model and obtain its performance:
 - A) model: $f_k(\mathbf{x}) = \sum_{j=1}^{N3} \alpha_{kj} x_j$, j belongs to the $N3$ metrics, α_{kj} s are obtained using CoDE.
 - B) training performance of model $f_k(\mathbf{x})$: P_k .

end for

- 3) **Output:** for a testing vector \mathbf{x} , compute the predicted value using Equ.(2)

IV. EXPERIMENTAL METHODOLOGIES

In this section, we detail our experimental data sets, the performance measure and implementation.

A. Datasets

In order to facilitate others to reproduce results, we use 41 data sets from 11 open-source projects in PROMISE repository [7], [22]. The characteristics of these experimental data sets are shown in Table I. The columns of 'faulty modules', 'range of defects', and 'total defects' respectively record the number of modules having defects (with the percentages of faulty modules in the subsequent brackets), the ranges of defect numbers in the corresponding data sets, and the total number of defects in all modules of the corresponding data sets. The metric number of these data sets is 20. Details of the specific twenty metrics can be found Jureczko et al. 's work [8].

B. Performance Measure

As mentioned in Yang et al.'s study [4], models with higher prediction accuracy (smaller average absolute errors or relative errors) might give a worse ranking, while fault-percentile-average (FPA) [10] takes into account both practical use and the whole ranking performance of prediction models, which is consistent with cumulative lift chart [23] for measuring a ranking [4]. Therefore, we adopt FPA as the model performance measure.

Considering n modules f_1, f_2, \dots, f_n , listed in increasing order of predicted defect number, k_i as the actual defect number in the module f_i , and $k = k_1 + k_2 + \dots + k_n$ as the total number of defects in all modules, the proportion of actual defects in the top m predicted modules to the whole defects is $\frac{1}{k} \sum_{i=n-m+1}^n k_i$. Then FPA is defined as follows [10]:

$$\frac{1}{n} \sum_{m=1}^n \frac{1}{k} \sum_{i=n-m+1}^n k_i$$

Larger FPA means better ranking performance.

C. Implementation

There are two main objectives in this paper:

- 1) investigating whether the idea of ensemble can improve Yang et al.'s LTR approach;
- 2) and comparing ensemble approaches for cross-version defect prediction for the ranking task.

According to the two objectives, we first compare the learning-to-rank-based ensemble method (LTRE) with the single LTR

TABLE I
EXPERIMENTAL DATA SETS

Datasets name	module number	faulty modules	range of defects	total defects
ant-1.3	125	20(16%)	[0,3]	33
ant-1.4	178	40(22.5%)	[0,3]	47
ant-1.5	293	32(10.9%)	[0,2]	35
ant-1.6	351	92(26.2%)	[0,10]	184
ant-1.7	745	166(22.3%)	[0,10]	338
lucene-2.0	195	91(46.7%)	[0,22]	268
lucene-2.2	247	144(58.3%)	[0,47]	414
lucene-2.4	340	203(59.7%)	[0,30]	632
xalan-2.4	723	110(15.2%)	[0,7]	156
xalan-2.5	803	387(48.2%)	[0,9]	531
xalan-2.6	885	411(46.4%)	[0,9]	625
xalan-2.7	909	898(98.8%)	[0,8]	1213
xerces-init	162	77(47.5%)	[0,11]	167
xerces-1.2	440	71(16.2%)	[0,4]	115
xerces-1.3	453	69(15.2%)	[0,30]	193
xerces-1.4	588	437(74.3%)	[0,62]	1596
camel-1.0	339	13(3.8%)	[0,2]	14
camel-1.2	608	216(35.5%)	[0,28]	522
camel-1.4	872	145(16.6%)	[0,17]	335
camel-1.6	965	188(19.5%)	[0,28]	500
ivy-1.1	111	63(56.8%)	[0,36]	233
ivy-1.4	241	16(6.6%)	[0,3]	18
ivy-2.0	352	40(11.4%)	[0,3]	56
synapse-1.0	157	16(10.2%)	[0,4]	21
synapse-1.1	222	60(27.0%)	[0,7]	99
synapse-1.2	256	86(33.6%)	[0,9]	145
velocity-1.4	196	147(75.0%)	[0,7]	210
velocity-1.5	214	142(66.4%)	[0,10]	331
velocity-1.6	229	78(34.1%)	[0,12]	190
jedit-3.2	272	90(33.1%)	[0,45]	382
jedit-4.0	306	74(24.2%)	[0,23]	226
jedit-4.1	312	79(25.3%)	[0,17]	217
jedit-4.2	367	48(13.1%)	[0,10]	106
jedit-4.3	492	11(2.2%)	[0,2]	12
log4j-1.0	135	34(25.2%)	[0,9]	61
log4j-1.1	109	37(33.9%)	[0,9]	86
log4j-1.2	205	189(92.2%)	[0,10]	498
poi-1.5	237	141(59.5%)	[0,20]	342
poi-2.0	314	37(11.8%)	[0,2]	39
poi-2.5	385	248(64.4%)	[0,11]	496
poi-3.0	442	281(63.6%)	[0,19]	500

approach. Subsequently, we conduct a comparison of six ensemble methods over 30 sets of cross-version data: LTRE, random forest [21], Bagging [24], Extra-Trees [25], gradient boosting [26], and Adaboost [27].

Because we use the same data sets as Yang and Wen's paper [7], we directly use their experimental results of both LTR and random forest in this paper. LTRE is implemented in Java. For LTRE, N1 is set to 100, N2 is set to half of all metrics, N3 is set to 2, C1 is set to 0.1, and W is set to 3. Bagging, Extra-Trees, gradient boosting, and Adaboost are implemented in Python, and their parameters are tuned by GridSearchCV method which provides the grid search that exhaustively generates candidates from a grid of parameter values specified in advance. For each method, only a small subset of parameters which play an important role are tuned, while others are set as their default values. Because the best parameters for different data sets are different, we choose the most frequently occurring ones. Bagging uses decision trees as base estimator, the number of trees is set to 200, and the

number of metrics is set to 0.4 of all metrics. To be noted, the number of metrics for Bagging is not all metrics (the performance of all metrics is worse than 0.4 of all metrics), so the Bagging here is not the traditional Bagging [11]. The number of trees in Extra-Trees, gradient boosting and Adaboost is also set to 200, the number of metrics in Extra-Trees and gradient boosting is set to 1/3 of all metrics, and the learning rate in gradient boosting and Adaboost is set to 0.1.

In order to simulate the actual situation, we adopt cross-version defect prediction, whose merit has been shown in Shukla et al.'s work [22]. SDP models constructed according to one version are used to predict defects of the next version, which is similar to actual use. All methods run 10 times for the same set of data. We use Wilcoxon rank-sum test [28] (which is called ranksum for short) to test whether 10 results by one method is significantly larger than those by another method using the same training and testing data.

V. EXPERIMENTAL RESULTS

In this section, we firstly show the comparison of the learning-to-rank-based ensemble approach (LTRE) and the single learning-to-rank approach (LTR) in order to see whether the idea of ensemble can improve LTR for SDP. Subsequently, the results of all compared ensemble approaches for cross-version defect prediction are presented.

A. Comparison of LTRE with LTR

In this subsection, we compare LTRE and LTR. Experimental results are shown in Table II. 'Ant-1.3-1.4' in Table II means using ant-1.3 as the training set and ant-1.4 as the testing set, and others are similar. Hence, there are totally 30 sets of data (each set includes one training version and one testing version). The columns of 'LTRE' and 'LTR' respectively record their mean FPA testing results, and the column of 'pvalues' records the p-values of ranksum test between 10 results achieved respectively by LTRE and LTR over the same set of data. We use bold type to mark the significantly larger results at the 0.05 significance level.

In Table II, '22' in the row of 'larger times' means that LTRE achieves larger mean FPA values than LTR over 22 sets of data. '18' in the row of 'significantly larger times' means that LTRE achieves significantly larger mean FPA values than LTR at the 0.05 significance level over 18 out of 30 sets of data. From these results, the idea of ensemble can benefit LTR for SDP for the ranking task.

B. Comparison of Six Ensemble Approaches

In this subsection, we show the experimental results of six ensemble approaches: LTRE, random forest (RF), Bagging, Extra-Trees (ET), gradient boosting (GB), and Adaboost (Ada). Table III shows the mean calculated over ten testing FPA results in ten runs for these methods. We use bold type to mark the largest results.

The best methods for different sets of data are very different: one method achieves best results over some sets of data, but

TABLE III
MEAN FPA RESULTS OF ALL METHODS FOR CROSS-VERSION DEFECT PREDICTION

Datasets	LTRE	RF	Bagging	ET	GB	Ada
ant-1.3-1.4	0.5921	0.5941	0.6043	0.6366	0.5734	0.5781
ant-1.4-1.5	0.7789	0.6956	0.6779	0.6325	0.5574	0.5809
ant-1.5-1.6	0.8247	0.7656	0.7708	0.7632	0.7107	0.7529
ant-1.6-1.7	0.8251	0.8226	0.8191	0.8085	0.7876	0.8203
lucene-2.0-2.2	0.7168	0.7132	0.6535	0.6656	0.6182	0.6467
lucene-2.2-2.4	0.6937	0.6560	0.7870	0.7809	0.7877	0.7604
xalan-2.4-2.5	0.6417	0.6226	0.7654	0.7707	0.7618	0.7591
xalan-2.5-2.6	0.6725	0.6835	0.7812	0.7833	0.7332	0.7716
xalan-2.6-2.7	0.5685	0.5703	0.7648	0.7486	0.6741	0.7781
xerces-init-1.2	0.4944	0.7470	0.8593	0.8632	0.8553	0.8507
xerces-1.2-1.3	0.5373	0.7159	0.8441	0.8358	0.8230	0.8120
xerces-1.3-1.4	0.7622	0.7275	0.8515	0.8470	0.8431	0.8718
camel-1.0-1.2	0.6735	0.6568	0.7760	0.7187	0.7371	0.6393
camel-1.2-1.4	0.7738	0.7891	0.7717	0.7841	0.7721	0.7781
camel-1.4-1.6	0.7461	0.7652	0.5541	0.5599	0.5430	0.5497
ivy-1.1-1.4	0.7716	0.7850	0.7109	0.7118	0.7116	0.6971
ivy-1.4-2.0	0.8244	0.7834	0.6490	0.6378	0.6364	0.6832
synapse-1.0-1.1	0.7187	0.7070	0.6940	0.6825	0.6570	0.6544
synapse-1.1-1.2	0.6874	0.7042	0.5017	0.5164	0.5137	0.4940
velocity-1.4-1.5	0.6008	0.6733	0.6813	0.6683	0.6572	0.6904
velocity-1.5-1.6	0.7233	0.7582	0.7007	0.6513	0.6527	0.6847
jedit-3.2-4.0	0.8453	0.8610	0.7056	0.6963	0.6772	0.6857
jedit-4.0-4.1	0.8190	0.8438	0.6651	0.6463	0.6509	0.6680
jedit-4.1-4.2	0.8727	0.8555	0.7551	0.7611	0.7611	0.7547
jedit-4.2-4.3	0.6529	0.7227	0.6185	0.5907	0.6149	0.6340
log4j-1.0-1.1	0.8140	0.7819	0.6772	0.6404	0.6803	0.6749
log4j-1.1-1.2	0.5594	0.5577	0.5652	0.5592	0.5653	0.5640
poi-1.5-2.0	0.7003	0.6968	0.7360	0.7248	0.7306	0.7227
poi-2.0-2.5	0.5811	0.5020	0.6985	0.6998	0.6575	0.6726
poi-2.5-3.0	0.6854	0.6785	0.7211	0.7133	0.6970	0.7000
maximum times	8	8	4	5	2	3

worst results over other sets of data. For example, LTRE achieves best result over ant-1.4-1.5, but worst result over xerces-init-1.2. As a whole, LTRE and random forest achieve maximum mean FPA values in most sets of data (eight for each method). These results imply that LTRE is comparable with other ensemble methods.

Bagging achieves larger mean results than Extra-Trees, gradient boosting, and Adaboost over more data sets. In addition, Extra-Trees and Adaboost perform better than gradient boosting over more data sets. To be noted, as mentioned in Section IV, the number of metrics for Bagging is set to 40% of all metrics because the performance of all metrics is worse than partial metrics. And thus the Bagging here is actually a combination of bagging and random subspace. This implies that random subspace might be better than boosting methods for these data sets.

VI. THREATS TO VALIDITY

In this paper, we investigate cross-version defect prediction based on 41 data sets from 11 projects in PROMISE repository, which reflects that the obtained results are strongly related to the SDP domain. However, there are some threats that may have an effect on our experimental results.

One threat is that only a few parameters of all experimental approaches are tuned in a small range, and the results might be different with other parameters setting.

In addition, the experimental data sets are only a very small part of all data sets, among which there are many data sets

based on industrial software systems [12] that are not publicly available. Other data sets might include different metrics and totally different modules. The conclusions over these data sets might not hold for other data sets.

The compared ensemble methods are only a small part of all ensemble methods. We use decision tree regression as the base learner for the compared ensemble methods in this paper. When other base learners are used, the conclusion might be different.

VII. CONCLUSIONS

Many studies have been conducted to sort software modules according to the number of defects [5], [10]. Random forest performed best in some previous studies [4], [10]. The good performance of random forest for SDP implies that the idea of ensemble might be useful to improve the performance of defect ranking models. Yang et al. proposed a learning-to-rank method (LTR) particularly for SDP [4], and their experimental results showed its effectiveness. Therefore, we propose a novel ensemble method based on LTR, which is called the learning-to-rank-based ensemble method (LTRE), in order to see whether the idea of ensemble can also improve LTR for sorting module in order of defect count.

Experimental results over 30 sets of cross-version data show that the proposed LTRE perform better than LTR, which implies that the idea of ensemble can benefit LTR for SDP.

In addition, we conduct a comparison study of six ensemble approaches for cross-version defect prediction for the ranking

TABLE II
MEAN FPA RESULTS OF LTRE AND LTR FOR CROSS-VERSION DEFECT
PREDICTION

Datasets	LTRE	LTR	pvalues
ant-1.3-1.4	0.5921	0.5917	0.6771
ant-1.4-1.5	0.7789	0.7113	0.0002
ant-1.5-1.6	0.8247	0.7331	0.0002
ant-1.6-1.7	0.8251	0.8176	0.0002
lucene-2.0-2.2	0.7168	0.7125	0.0028
lucene-2.2-2.4	0.6937	0.6845	0.0002
xalan-2.4-2.5	0.6417	0.6467	0.0002
xalan-2.5-2.6	0.6725	0.6645	0.0006
xalan-2.6-2.7	0.5685	0.5692	0.1212
xerces-init-1.2	0.4944	0.6266	0.0002
xerces-1.2-1.3	0.5373	0.6763	0.0002
xerces-1.3-1.4	0.7622	0.7190	0.0002
camel-1.0-1.2	0.6735	0.6145	0.0002
camel-1.2-1.4	0.7738	0.7541	0.0002
camel-1.4-1.6	0.7461	0.7451	0.2263
ivy-1.1-1.4	0.7716	0.7489	0.0002
ivy-1.4-2.0	0.8244	0.7949	0.0002
synapse-1.0-1.1	0.7187	0.6661	0.0002
synapse-1.1-1.2	0.6874	0.6305	0.0002
velocity-1.4-1.5	0.6008	0.6153	0.0017
velocity-1.5-1.6	0.7233	0.7495	0.0002
jedit-3.2-4.0	0.8453	0.8447	0.4274
jedit-4.0-4.1	0.8190	0.8244	0.0046
jedit-4.1-4.2	0.8727	0.8692	0.0028
jedit-4.2-4.3	0.6529	0.6900	0.0002
log4j-1.0-1.1	0.8140	0.7996	0.0003
log4j-1.1-1.2	0.5594	0.5561	0.1212
poi-1.5-2.0	0.7003	0.6860	0.0028
poi-2.0-2.5	0.5811	0.5131	0.0002
poi-2.5-3.0	0.6854	0.6526	0.0002
larger times	22	8	
significantly larger times	18	7	

task over 41 publicly available data sets. The comparison results show that LTRE can perform comparably with other ensemble methods. LTRE and random forest achieve maximum mean FPA values in most sets of data. Nevertheless, the compared ensemble methods have their own advantages in sorting modules in order of defect count over the experimental data sets. The best methods for different sets of data are very different. Therefore, in our future work, we are going to investigate how to choose the appropriate method according to the characteristics of data.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China (Grants No. 61602534) and Fundamental Research Funds for the Central Universities (No.17lgpy122).

REFERENCES

- [1] T. Ostrand, E. Weyuker, and R. Bell, "Predicting the location and number of faults in large software systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340–355, 2005.
- [2] B. Boehm and V. Basili, "Software defect reduction top 10 list," *IEEE Computer*, vol. 34, pp. 135–137, 2001.
- [3] Z. Xu, J. Liu, X. Luo, and T. Zhang, "Cross-version defect prediction via hybrid active learning with kernel principal component analysis," in *25th IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2018, pp. 209–220.
- [4] X. Yang, K. Tang, and X. Yao, "A learning-to-rank approach to software defect prediction," *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 234–246, 2015.
- [5] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, pp. 1–47, 2011.
- [6] J. Nam and S. Kim, "Clami: defect prediction on unlabeled datasets," in *30th IEEE/ACM International Conference on Automated Software Engineering*, 2015. IEEE/ACM, 2015, pp. 452–463.
- [7] X. Yang and W. Wen, "Ridge and lasso regression models for cross-version defect prediction," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 885–896, 2018.
- [8] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10, 2010, pp. 9:1–9:10.
- [9] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *IEEE Transactions on Software Engineering*, vol. 22, no. 12, pp. 886–894, 1996.
- [10] E. Weyuker, T. Ostrand, and R. Bell, "Comparing the effectiveness of several modeling methods for fault prediction," *Empirical Software Engineering*, vol. 15, no. 3, pp. 277–295, 2010.
- [11] S. Rathore and S. Kumar, "Ensemble methods for the prediction of number of faults: a study on eclipse project," in *11th International Conference on Industrial and Information Systems*, 2016, pp. 540–545.
- [12] K. Gao and T. Khoshgoftaar, "A comprehensive empirical study of count models for software defect prediction," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 223–236, 2007.
- [13] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 37, no. 3, pp. 356–370, 2011.
- [14] L. Kumar, S. Rath, and A. Sureka, "An empirical analysis on effective fault prediction model developed using ensemble methods," in *41th Annual Computer Software and Applications Conference*, 2017, pp. 244–249.
- [15] C. Yohannese, T. Li, M. Simfukwe, and F. Khurshid, "Ensembles based combined learning for improved software fault prediction: a comparative study," in *12th International Conference on Intelligent Systems and Knowledge Engineering*, 2017, pp. 1–6.
- [16] S. Huda, K. Liu, M. Abdelrazek, A. Ibrahim, S. Alyahya, H. Al-Dossari, and S. Ahmad, "An ensemble oversampling model for class imbalance problem in software defect prediction," *accepted by IEEE Access*, 2018.
- [17] Z. Li, X. Jing, X. Zhu, and H. Zhang, "Heterogeneous defect prediction through multiple kernel learning and ensemble learning," in *2017 International Conference on Software Maintenance and Evolution*, 2017, pp. 91–102.
- [18] Z. Sun, Q. Song, and X. Zhu, "Using coding-based ensemble learning to improve software defect prediction," *IEEE Transactions on Systems, Man and Cybernetics-Part C: Applications and Reviews*, vol. 42, no. 6, pp. 1806–1817, 2012.
- [19] X. Yu, J. Liu, Z. Yang, X. Jia, Q. Ling, and S. Ye, "Learning from imbalanced data for predicting the number of software defects," in *28th International Symposium on Software Reliability Engineering*, 2017, pp. 78–89.
- [20] Y. Wang, Z. Cai, and Q. Zhang, "Differential evolution with composite trial vector generation strategies and control parameters," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 55–66, 2011.
- [21] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [22] S. Shukla, T. Radhakrishnan, and K. Muthukumaran, "Multi-objective cross-version defect prediction," *Soft Computing*, pp. 1–22, 2016.
- [23] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 561–595, 2008.
- [24] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.
- [25] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.
- [26] J. Friedman, "Greedy function approximation: a gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.
- [27] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, pp. 119–139, 1997.
- [28] M. Fay and M. Proschan, "Wilcoxon-mann-whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules," *Statistics Surveys*, vol. 4, pp. 1–39, 2010.

A Multilevel Analysis Method for Architecture Erosion

Tong Wang, Dongdong Wang, Bixin Li
School of Computer Science and Engineering
Southeast University, Nanjing, China

Abstract—During the evolution of software, improper change operations may cause architecture erosion. Architecture erosion creates problems on evolutionary costs, software performance and software quality. Many methods have been proposed to analyze architecture erosion. Architecture depends on the implementation of code, that is, architecture erosion is caused by code. However, few methods analyze the reasons for architecture erosion based on code. Besides, architecture is eroded with software evolution, but most methods do not analyze architecture erosion based on the change of software. In this paper, we propose a multilevel analysis method for architecture erosion. Our method contains three steps. Firstly, we detect the changed pairs based on two architectures by performing a multilevel change detection method. Secondly, we detect whether the corresponding code elements of changed pairs are erosion points by calculating erosion degree. Thirdly, we establish a cost-benefit model of repairing architecture erosion for repairing architecture erosion more effectively with repaired few erosion points. We illustrate our method through an open source project, and the experimental results indicate that our method can detect the erosion points of each level and the cost-benefit model is effectively.

Index Terms—architecture erosion, multilevel change detection, the cost-benefit model

I. INTRODUCTION

Software is continually evolved to meet new requirements. With the evolution of software, architecture erosion often occurs [1], [2]. Architecture erosion creates problems such as the increase of software evolutionary costs [3], the decrease of software performance [4], and the degradation of software quality [5]. In a word, uncontrolled architecture erosion has a negative impact on software [6]. In order to reduce the negative impacts, analyzing software architecture erosion in time is an important task during the development and maintenance of software [7].

Some researchers propose analysis methods for architecture erosion based on multiple types of indicators. Zhang et al. detect architecture erosion based on the design decision of architectural pattern [8], and Herold et al. detect architecture erosion based on a common ontology [7]. The purpose of detecting and repairing architecture erosion is reducing the effects on architecture quality [9], that is, the decrease of architecture quality is an important sign of architecture erosion, but few methods analyze architecture erosion based on quality. Besides, architecture depends on the implementation of source code [10], but few methods analyze architecture erosion based

on code. Also, there may be many erosion points, but it is impossible to repair all erosion points, so how to reduce the erosion degree of architecture effectively by repairing fewer erosion points is an important problem.

To overcome the above limitations, we proposed the multilevel analysis method for architecture erosion. In our method, we first detect the changed pairs of each level by performing the multilevel change detection method. Then, we detect whether the corresponding code elements of changed pairs are erosion points by calculating the erosion degree. Finally, we establish the cost-benefit model for providing the repairing priority of erosion points.

To sum up, this work makes the following contributions.

- We detect the erosion points based on architecture quality which is an important sign of architecture erosion.
- We analyze reasons for architecture erosion based on the implementation of architecture by using the multilevel change detection method.
- We establish the cost-benefit model of repairing architecture erosion, in order to more effective repairing architecture erosion by repairing fewer erosion points.

This paper is organized as follows. Section II introduces the details of our method. In section III, we illustrate our method though an open source project. Section IV presents related work. In Section V, the threats to the validity of our method are analyzed. In Section VI, we make a conclusion and discuss future work.

II. OUR METHOD

A. Method Overview

Architecture erosion leads to the gradual deterioration of software quality [11], that is, the deterioration of quality is an important sign of architecture erosion. So, in this paper, we define architecture erosion as follows.

Definition 1 architecture erosion: It is a phenomenon that occurs when architecture quality is decreased with software evolution.

The component dependency graph is a widely acceptable representation of architecture, in our method, we use it to represent architecture. The component dependency graph consists of the component and the dependency between components. A component is a code element of the high abstraction level, and it consists of multiple files. A file consists of multiple statements. According to the granularity of code elements, we

B. Li is the corresponding author. E-mail: bx.li@seu.edu.cn
DOI reference number:10.18293/SEKE2019-046

divide architecture into three levels, component level, file level, and statement level.

The architecture contains multiple components. In an eroded architecture, not all components are eroded. Similarly, in an eroded component, not all files are eroded. And in an eroded file, not all statements are eroded. In order to narrow down the range of erosion for reducing the repairing cost, we analyze architecture erosion from architecture to component level, then from component level to file level, finally from file level to statement level.

Architecture is eroded with software evolution, so does components, files, and statements. In order to analyze whether components, files, and statements are eroded, we should first detect changed pairs of each level by using a multilevel change detection method. After detecting the changed pairs, we detect whether the corresponding code of changed pairs are eroded.

Definition 2 changed pair: Let $A1$ be a original architecture. Let $A2$ be a current architecture. Let $C1$ and $C2$ be two code elements of the above architectures. If $C1$ is evolved to $C2$, the two code elements constitute a changed pair. If $C1$ is deleted and $C2$ is a new code element, then they respectively constitute changed pairs with a virtual code element.

According to the above analysis, the overview of our method is shown in Figure 1. The first step is detecting the changed pairs of code elements, where add, delete and update operations are denoted in the green, red and blue border. The second step is calculating erosion degree to detect whether the corresponding code elements of changed pairs are erosion points, and the erosion points are denoted in gray nodes. The third step is establishing the relations between the number of repaired erosion points with the degree of decline in architecture erosion, that is, we establish a cost-benefit model of repairing erosion points. The model demonstrates how to reduce more erosion degree of architecture by repairing fewer erosion points.

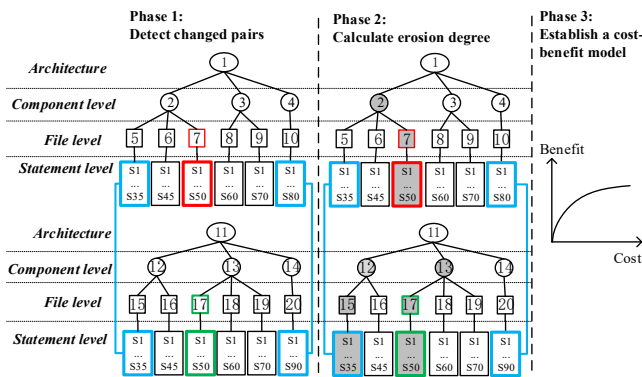


Fig. 1. An overview of our method.

B. Detecting changed pairs of each level

In order to detect changed pairs at file level and statement level, we perform the change detection method named the two-step multilevel program analysis tree matching method [12].

The method can detect multilevel changes effectively, such as method level, class level, and file level.

We perform the following steps to detect changed pairs at component level: calculating the similarity between components and identifying change operation based on component similarity.

The component similarity indicates the degree of similarity. The component consists of multiple files, so we calculate the component similarity based on the contained files. The formula is shown in Formula 1.

$$Sim(i, j) = \frac{2 * |S_i \cap S_j|}{L_i + L_j} \quad (1)$$

where $Sim(i, j)$ is the similarity between the i_{th} component and the j_{th} component, S_i is the file set contained in the i_{th} component, $|X|$ is the number of elements of the set X and L_i is the number of code lines.

The components of the original architecture and the components of the current architecture are matched based on the component similarity to detect which components constitute a changed pair. We consider the following three change operations at the component level.

Add(C): A new component C is added in V_2 .

Delete(C): In V_2 , the component C is deleted.

Update(C_i, C_j): The component C_i is updated to C_j .

In order to identify the above change operations based on the component similarity, we define two thresholds. The first threshold is *HighThreshold*. If the component similarity above *HighThreshold*, we deem that the two components are the same, that is, the component is not changed during the evolution process. The second threshold is *LowThreshold*. If the component similarity above *HighThreshold* but below *LowThreshold*, we deem that the change operation between the changed pair is updating. If the component similarity below *LowThreshold*, we deem that the two components of the changes pair are respectively the new component of the current architecture and the deleted component of the original architecture. The algorithm of identifying change operations is shown in Algorithm 1.

C. Calculating erosion degree

In this step, we detect whether the code elements of each changed pair are eroded, that is, whether the code elements are the erosion points.

Definition 3 erosion point: Let A_i and A_j constitute a changed pair which respectively belong to the original architecture and the current architecture. If the change operation between A_i and A_j causes architecture erosion, A_i and A_j are the erosion points.

The decrease in architecture quality is an important sign of architecture erosion. When architecture quality is decreased, it indicates that architecture is eroded, correspondingly, if architecture quality is increased, architecture is nor eroded.

In this paper, we take the change of understandability and testability as the indicators of architecture erosion. Understandability and testability are important quality attributes

Algorithm 1 The algorithm of identifying change operations**Input:** The component list of the original version *oriCom*The component list of the current version *curCom*The threshold of similarity *threshold***Output:** The list of the adding change *add*The list of the deleting change *delete*The list of the updating change *update*

```

1: Let oriMatched store the matched component of oriCom
2: Let curMatched store the matched component of curCom
3: Let match mark the component whether has been matched successfully
4: Set match = false
5: for each oriComponent ∈ oriCom do
6:   if match = true then
7:     match = false
8:     Continue
9:   end if
10:  for each curComponent ∈ curCom do
11:    Let similarity is the similarity between oriComponent and curComponent
12:    if similarity = 1 then
13:      put oriComponent in oriMatched
14:      put curComponent in curMatched
15:      match = true Break
16:    else
17:      if similarity > threshold then
18:        put oriComponent in oriMatched
19:        put curComponent in curMatched
20:        put < oriComponent, curComponent > in update
21:        match = true Break
22:      end if
23:    end if
24:  end for
25: end for
26: delete = oriCom - oriMatched
27: add = curCom - curMatched

```

for architecture [13]. Firstly, we measure the two quality attributes.

Testability determines the required resources during testing and making test plans [14]. Its calculating formula is shown in Formula 2.

$$Testability = 1 - \frac{\sum_{i=1}^N \frac{O_i}{N}}{N} \quad (2)$$

where T is the measurement of testability, N is the number of elements, i denotes the i_{th} code element, O is the fanout of the i_{th} element.

Understandability denotes the difficulty of understanding architecture [15]. Its calculating formula is shown in Formula 3.

$$Understandability = \frac{\sum_{i=1}^N f(C_i)}{N} \quad (3)$$

where U is the measurement of understandability, N is the number of elements, i denotes the i_{th} element, L_i is number of code lines of the i_{th} element, $f(C_i)$ is the density of cognitive complexity of the code element C_i . The calculating formula of the density of cognitive complexity is shown in Formula 4.

$$f(C_i) = \begin{cases} 1 & \frac{C_i}{L_i} \leq 0.05 \\ 0.8 & 0.05 < \frac{C_i}{L_i} \leq 0.1 \\ 0.6 & 0.1 < \frac{C_i}{L_i} \leq 0.2 \\ 0.4 & 0.2 < \frac{C_i}{L_i} \leq 0.3 \\ 0.2 & 0.3 < \frac{C_i}{L_i} \leq 0.4 \end{cases} \quad (4)$$

where $f(C_i)$ is the density of cognitive complexity of the code element C_i , L_i is number of code lines of the i_{th} element and C_i is the cognitive complexity of the i_{th} element. Cognitive complexity can yield assessments of control flow that correspond to programmers intuitions about the mental, or cognitive effort required to understand those flows.

According to the above formulas, we the measurements of understandability and testability, then we use the erosion degree to represent the change of architecture quality before and after evolution.

Definition 4 erosion degree: Let A_o and A_c be architectures before and after software evolution. Let $A_o[i]$ be a code element of A_o and $A_c[j]$ be a code element of A_c . If architecture quality is decreased, the difference between $A_o[i]$ and $A_c[j]$ is the erosion degree. If erosion degree is a negative value, it indicates that the changed pair does not damage architecture quality instead of increasing architecture quality.

The erosion degree is calculated based on a pair of code elements before and after software evolution. To architecture, the erosion degree depends on the change of the contained components. Similarity, to a component, the erosion degree depends on the changes of the contained files. Then, to file, the erosion degree depends on the changes of the contained statements.

The elements of the pair respectively belong to the original architecture and the current architecture. When a code element is deleted from the original architecture, and when a code element is added in the current architecture, there is not a corresponding code element in another architecture, so we use the average measurement as the threshold to calculate the erosion degree of the new code element and the deleted code element.

The formula of erosion degree is shown in Formula 5.

$$f(E_i, E_j) = \begin{cases} M_j - M_i & E_i \text{ is evolved to } E_j \\ M_j - AVG & E_j \text{ is a new code element} \\ AVG - M_i & E_i \text{ is a deleted code element} \end{cases} \quad (5)$$

where E_i is the i_{th} code element, $f(E_i, E_j)$ is difference between the measurements of E_i and E_j , M_i is the measurement of the i_{th} code element and AVG is the average measurement of the previous architecture.

D. Establishing the cost-benefit model

Considering the number of erosion points and the repairing cost, in actual development process, it is impossible to repair all erosion points. In order to more effective repair architecture by repairing fewer erosion points, we establish the cost-benefit model based on erosion contribution.

Definition 5 erosion contribution: Let C be an erosion point. The erosion contribution of C denotes that, compared with other eroded code elements of the same level, the degree of influence on architecture erosion.

The formula of the erosion contribution is shown in Formula 6.

$$EC(P_i, C_j) = \frac{f(P_i, C_i)}{\sum_{(a,b) \in S} f(P_a, C_b)} * PEC(P_i, C_i) \quad (6)$$

where O_i is the i_{th} code element of the original architecture, C_i is the i_{th} element of the current architecture, $EC(O_i, C_j)$ is the erosion contribution of the changed pair constituted by O_i and C_j , $f(O_i, C_j)$ is the erosion degree of the changed pair constituted by O_i and C_j , S is the set of changed pairs which are erosion points, $|S|$ is the number of the elements of S , (a, b) is the element of S and it denotes the changed pair constituted by O_a and C_b , and $PEC(P_i, C_i)$ is the erosion contribution of the changed pair which is the corresponding higher abstraction elements of O_i and C_j . The component level is the highest abstraction level in our method. If O_i is a component, the value of $PEC(O_i, C_i)$ is 1.

The higher erosion contribution indicates that the corresponding point has caused more negative effects on architecture. So the erosion contribution denotes the relative urgency degree of repairing. The order of repairing erosion points is based on the erosion contribution from high to low.

Unlike the erosion degree, the erosion contribution is a comparative value, because it is calculated based on the erosion of other code elements of the same level. The sum of erosion contribution of all erosion points is 1. The sum of the erosion contribution of all repaired erosion points denotes the degree of repairing architecture erosion, so it is used as the indicator of the benefit.

Architecture depends on the implementation of the code, and in the actual development process, developers repair architecture erosion by modifying the code. So in our model, we only take the erosion points belonging to statement level into consideration.

Lines of code (LOC) is an important indicator for estimating software costs, so we use the number of repaired erosion points as the cost. With repairing erosion points, the erosion degree of architecture is reduced which is caused by repairing erosion points, so we use the sum of erosion contribution of the repaired erosion points as the benefit.

The erosion contribution denotes the degree of effects on architecture erosion. We sort all erosion points according to the erosion contribution from high to low. There is a relation between the number of repaired erosion points and the sum of erosion contribution, and the cost-benefit model is established based on the above relation.

III. ILLUSTRATIVE EXAMPLE

In this section, we choose an open source project as the case to illustrate the process of our method. *commons-lang* is a package which provides some basic APIs for some general

operations, such as automatically generating *toString()* results, automatically implementing *hashCode()* and *equals()*, etc. *commons-lang* has received 1.5K stars in *GitHub*.

We choose four versions as cases to detect architecture erosion. There is not a document about the architecture of *commons-lang*, so we perform the recovery method proposed by Kong et al. [16] to obtain its architecture. The basic information of *commons-lang* is shown in Table I, where the first column is the version number, the second column is the number of code lines, the third column is the measurement of testability, and the fourth column is the measurement of understandability.

TABLE I
THE BASIC INFORMATION OF COMMONS-LANG

Version	LOC	Testability	Understandability
3.1.0	52.8K	0.800	0.760
3.2.0	61.6K	0.733	0.667
3.3.0	63.5K	0.834	0.667
3.4.0	66.0K	0.834	0.760

Table I shows that, compared with the measurements of version 3.1.0, the measurements of version 3.2.0 are decreased. The decreasing measurements indicate that the architecture is eroded in the evolution process, so we analyze the evolution process. The erosion degree of understandability is -0.093, and the erosion degree of testability is -0.067.

The first step is detecting changed pairs. We implement the multilevel change detection method to detect the changed pairs of file level and statement level, and detect changed pairs of component level based on the component similarity. The number of changed code elements of each level is shown in Table II, where the first column is the change operation, where the second column to the fourth column are respectively component level, file level, and statement level. As the table shows that, there are six changed pairs at the component level, 220 changed pairs at the file level, and 6829 changed pairs at the statement level.

TABLE II
THE NUMBER OF CHANGED PAIRS OF EACH LEVEL

Type	The number of changed code elements		
	Component level	File level	Statement level
Add	1	29	3416
Delete	3	0	822
Update	2	191	1798
Move	\	0	803
Total	6	220	6839

The second step is calculating erosion degree of each changed pairs. We first calculate the erosion degree of changed pairs of the component level, and the values of the erosion degree are shown in Table III, where the first column is the change operation of the component level, the second column is the erosion degree to testability, and the third column is the erosion degree to understandability. In the first column, we

use *update*, *add* and *delete* to represent the change operation, and the corresponding element is the changed component. As shown in Table III, *Add(src-1)* has the highest erosion degree of testability, and *Delete(common\lang3-2)* has the highest erosion degree of understandability.

TABLE III
THE EROSION DEGREE OF CHANGED PAIRS OF THE COMPONENT LEVEL

Change operation	ED_Test.	ED_Under.
Update(common\lang3)	2.380	-2.400
Update(src)	-0.571	-5.700
Delete(common\lang3-1)	0.142	4.100
Delete(common\lang3-2)	-0.570	3.950
Delete(lang3\text)	-0.714	1.850
Add(src-1)	0.333	-0.800

There are 220 changed pairs of file level and 6839 changed pairs of statement level. Due to the limitation of space, we only show the erosion degree of some changed pairs in Table IV, where the first column is the change operation of file level, the second column is the erosion degree of testability, and the third column is the erosion degree of understandability. As shown in Table IV, *Update(TypeUtils.java)* has the highest erosion contribution to testability, *Add(Conversion.java)* has the highest erosion contribution to understandability.

TABLE IV
THE EROSION DEGREE OF CHANGED PAIRS OF FILE LEVEL

Change operation	ED_Test.	ED_Under.
Add(Conversion.java)	0	0.363
Add(InheritanceUtils.java)	0	0.233
Update(CharSequenceUtils.java)	0	0.215
Update(StrSubstitutor.java)	0	0.114
Update(FastDateFormatTest.java)	0	0.112
Update(TypeUtils.java)	0.376	0
Update(FastDateFormat.java)	0.233	0
ADD(TypeLiteral.java)	0.108	-0.057
ADD(FastDatePrinter.java)	0.108	0.087
ADD(Triple.java)	0.108	0.028
ADD(NotImplementedExceptionTest.java)	0.067	-0.086
Update(StringUtilsEqualsIndexOfTest.java)	0	-0.009

The third step is establishing the cost-benefit model based on erosion contribution. There are 6839 changed pairs of statement level, according to the erosion degree of them, we find that, 1027 changed pairs are erosion points for testability and 1453 changed pairs are erosion points for understandability. We sort these erosion points according to the erosion contribution from high to low. We respectively numbered them from 1 to 1027, and from 1 to 1453, then we establish the cost-benefit model. Due to the limitation of space, we only demonstrate the cost-benefit model of testability in Figure 2, where the x-axis represents the number of repaired erosion points, and the y-axis represents the sum of erosion contribution of the corresponding repaired erosion points.

As Figure 2 shows that, the growth rate of the sum of erosion contribution declines with repairing more erosion points, that is, the average benefit is reduced gradually. In

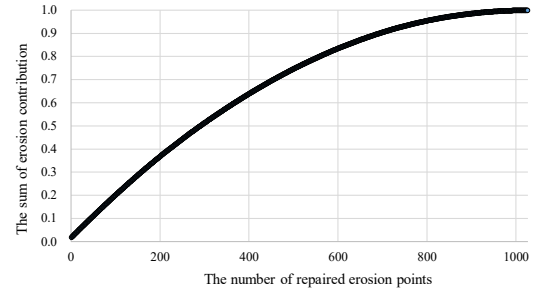


Fig. 2. The cost-benefit model of testability.

the example, we follow the Pareto principle which states that 80% of the effects come from 20% of the causes, so we only repair the top 20% erosion points.

We repair erosion points by rolling back them to their corresponding version. After rolling back the architecture is denoted as the new architecture. We detect erosion degree again based on the 3.1.0 version and the new architecture. The comparison of architecture quality is shown in Figure 3.

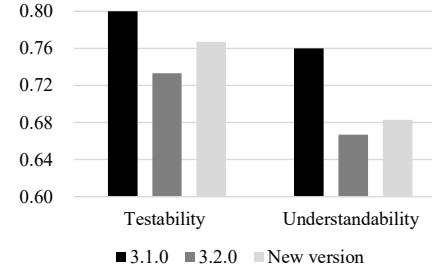


Fig. 3. The comparison of architecture quality.

According to Figure 3, we can draw two conclusions. First, compared with the version 3.2.0, the contribution degree is decreased, so the erosion points are detected and repaired. Second, only 20% erosion points are repaired, but the erosion degree falls by over 25%, so the cost-benefit model proposes the repairing priority effectively.

IV. RELATED WORK

The purpose of analyzing architecture erosion is how to locate the erosion points and how to repair the erosion points [10], so we compare our method with related work from the above aspects.

In the aspect of locating the erosion points, most researches propose analysis methods for architecture erosion based on one certain version. There are two main types of indicators, the inconsistency between the requirements and the actual implementation and the improper architecture pattern. Medvidovic et al. use the inconsistency as the indicator [17]. Zhang et al. detect architecture erosion of architectural pattern by the design decision [8]. Herold et al. detect architecture erosion of architecture pattern by the common ontology [7]. However, the above effects of architecture erosion all threaten the architecture quality [7], that is, avoiding the effects on

software quality is the ultimate goal. So, in our method, we use the architecture quality as the indicators to detect architecture erosion.

In the aspect of repairing the erosion points. Terra et al. provide an approach to provide recommendations for removing architectural violations detected by the dependency constraint language [10]. Mair et al. propose a heuristic search method for adequate repairs using formalized and explicit knowledge of software engineers [9]. However, the actual architecture is implemented by its source code [10], so architecture erosion need be repaired by modifying code. So, our method analyzes architecture erosion from multiple levels.

In addition to the above aspects, our method establishes the cost-benefit model for repairing architecture erosion. In order to provide the repairing priority.

V. THREATS TO VALIDITY

Construct validity. We choose the component dependency graph as the representation of software architecture. Due to the limitations of the documents of architecture, we obtain architecture by implementing an architecture recovery method. There may be some deviations caused by the limitations of the recovery method. However, the recovery method has high accuracy, and in the future, the recovery method can be replaced by other better methods.

Internal validity. In our example, we roll back erosion points to their previous version. The experimental results indicate that, after rolling back, architecture erosion is repaired. However, rolling back erosion points may be not effective for maintaining functionality. However, the cost-benefit model is still useful for providing the priority of repairing erosion points.

External validity. In this paper, we choose testability and understandability as the indicators to detect architecture erosion. The two quality are quantified based on the attributes of components and dependencies, so we can use other quality as the indicators if it is quantified based on the features of components and dependencies.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose a multilevel method for analyzing architecture erosion. Firstly, we detect changed pairs of each level by calculating component similarity and performing a multilevel change detection method. In this step, the changed code elements are matched. Secondly, we detect which changed pairs are the erosion points by calculating erosion degree. Thirdly, we establish the cost-benefit model of repairing architecture erosion based on the erosion contribution of each erosion point. The cost-benefit model is useful to obtain more benefits with less cost. We illustrate the process of our method through an example, and the experimental results indicate that our method can detect erosion points and the cost-benefit model is useful for more effective repairing architecture erosion by repairing fewer erosion points.

In future work, we will investigate how to propose more suggestions for repairing architecture instead of rolling back.

ACKNOWLEDGEMENTS

This work is supported in part by the National Key R&D Program of China under Grant 2018YFB1003902, in part by the Cooperation Project with Huawei Technologies Co., Ltd., under Grant YBN2016020009, and in part by National Natural Science Foundation of China under Grant 61872078, Grant 61572126, and Grant 61402103.

REFERENCES

- [1] David Faitelson, Robert Heinrich, and Shmuel Tyszbewicz. Supporting software architecture evolution by functional decomposition. In *International Conference on Model-Driven Engineering and Software Development*, pages 435–442, 2017.
- [2] Pooyan Behnamghader, Duc Minh Le, Joshua Garcia, Daniel Link, Arman Shahbazian, and Nenad Medvidovic. A large-scale study of architectural evolution in open-source software systems. *Empirical Software Engineering*, 22(3):1–48, 2016.
- [3] Hongyu Pei Breivold and Ivica Crnkovic. A systematic review on architecting for software evolvability. In *Australian Software Engineering Conference*, pages 13–22, 2010.
- [4] Fabian Brosig, Philipp Meier, Steffen Becker, Anne Kozirolek, Heiko Kozirolek, and Samuel Kounev. Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures. *IEEE Transactions on Software Engineering*, 41(2):157–175, 2015.
- [5] Ricardo Britto, Darja Smite, and Lars Ola Damm. Software architects in large-scale distributed projects: An ericsson case study. *IEEE Software*, 33(6):48–55, 2016.
- [6] Santonu Sarkar, Shubha Ramachandran, G. Sathish Kumar, Madhu K. Iyengar, K. Rangarajan, and Saravanan Sivagnanam. Modularization of a large-scale business application: A case study. *IEEE Software*, 26(2):28–35, 2009.
- [7] S Herold and A Rausch. Complementing model-driven development for the detection of software architecture erosion. In *Proceedings of the 5th International Workshop on Modeling in Software Engineering*, pages 24–30, 2013.
- [8] Hui Song Franck Chauvel Hong Mei Lei Zhang, Yanchun Sun. Detecting architecture erosion by design decision of architectural pattern. In *SEKE*, pages 758–763, 2011.
- [9] Matthias Mair and Sebastian Herold. Towards extensive software architecture erosion repairs. In *European Conference on Software Architecture*, pages 299–306, 2013.
- [10] Ricardo Terra, M. T. Valente, Krzysztof Czarnecki, and R. S. Bigonha. Recommending refactorings to reverse software architecture erosion. In *European Conference on Software Maintenance & Reengineering*, pages 335–340, 2012.
- [11] Lakshitha De Silva and Dharini Balasubramaniam. Controlling software architecture erosion: A survey. *Journal of Systems & Software*, 85(1):132–151, 2012.
- [12] Wang Tong, Wang Dongdong, Zhou Ying, and Li Bixin. Software multiple-level change detection based on two-step mpat matching. In *IEEE International Conference on Software Analysis, Evolution and Reengineering*, pages 4–14, 2019.
- [13] Srdjan Stevanetic and Uwe Zdun. Exploring the relationships between the understandability of components in architectural component models and component level metrics. In *International Conference on Evaluation & Assessment in Software Engineering*, pages 1–10, 2014.
- [14] Samar Mouchawrab, Lionel C Briand, and Yvan Labiche. A measurement framework for object-oriented software testability. *Information and software technology*, 47(15):979–997, 2005.
- [15] Jitender Kumar Chhabra, KK Aggarwal, and Yogesh Singh. Code and data spatial complexity: two important software understandability measures. *Information and software Technology*, 45(8):539–546, 2003.
- [16] Xianglong Kong, Bixin Li, Lulu Wang, and Wensheng Wu. Directory-based dependency processing for software architecture recovery. *IEEE Access*, 6:52321–52335, 2018.
- [17] Gruenbacher P Medvidovic N, Egyed A. Stemming architectural erosion by coupling architectural discovery and recovery. In *Proceedings of the 2nd International Software Requirements to Architectures Workshop*, pages 61–68, 2003.

The Affinity Platform:

Modular Architecture based on Independent Components

Alexandru Ardelean, Kuderna-Iulian Benta

Department of Computer Science

Faculty of Mathematics and Computer Science, Babeş-Bolyai University

Cluj-Napoca, Romania

ardeleanalexandru3@gmail.com and benta@cs.ubbcluj.ro

Abstract—This paper presents the Affinity Platform, a plugin platform for developing software systems centered around the idea of Independent Components. “Independent” refers to the ability of a plugin to run on a machine not just as part of a software system but also on its own. This way, software systems can be built out of chains of Independent Components, making it easier to partially reuse their Components and also to extend these systems in the future. The Affinity Platform uses a graphical user interface to represent a software system in the shape of a diagram. By analyzing this diagram, it can generate linking scripts for Components. The Affinity architecture also facilitates the development of distributed and parallel software systems, as Components are capable of operating independently on different threads and machines. The Affinity Platform was used in the development of an emotion recognition application in a multimodal setup running on a mobile and resource-scarce medium. In this scenario, the strongest points of the platform were the ability to offload code to other machines, the ease of adapting the system to different sensor setups, and the convenience of developing Components individually and only having to connect them at the end.

Keywords - CBSE, Independent Component, plugin architecture, platform, chaining.

I. INTRODUCTION

As software systems are becoming more open, modular, reusable, distributed, parallel, and context-aware, developers seem to have a hard time molding their software applications around all these requirements. Extending such systems is also continually becoming more challenging. And the list of must-have features is only getting longer. As the complexity of software systems rises above the level at which a manageable team can efficiently tackle it, new ways of organizing software systems must be developed.

Inspired by audio plugins based on the audio in/audio out paradigm, the Affinity architecture was envisioned for building data flows. The building blocks of the Affinity Platform are called Components [1]. A Component is abstracted as a data-processing unit with multiple channels for input and output and various configuration options. Components can be any software program capable of extracting, processing or delivering data over a medium, requiring at a minimum a channel for data input or a channel for data output. Components can be linked together into chained systems that can be automatically parallelized (on a

per-Component basis) or distributed among multiple machines. Locally available Components are connected via a piping mechanism, while connectivity between Components located on different machines is achieved by interpolating between them other Components responsible for sending and receiving data over the desired medium.

The scenario in which this architecture was tested is that of a context-aware software application for emotion detection. The data flow of the app starts with a Component running on a secondary device (A, a smartwatch) extracting data from multiple sensor setups and sending its output to a more powerful device (B, a smartphone). Device B runs a machine learning algorithm for emotion recognition that processes the data and sends the result over to a remote server (C, a cloud-based API). Device C is represented by a Component that computes a virtual diagram based on the data received from multiple A-B setups. A visual representation of this example is illustrated in Figure 1.

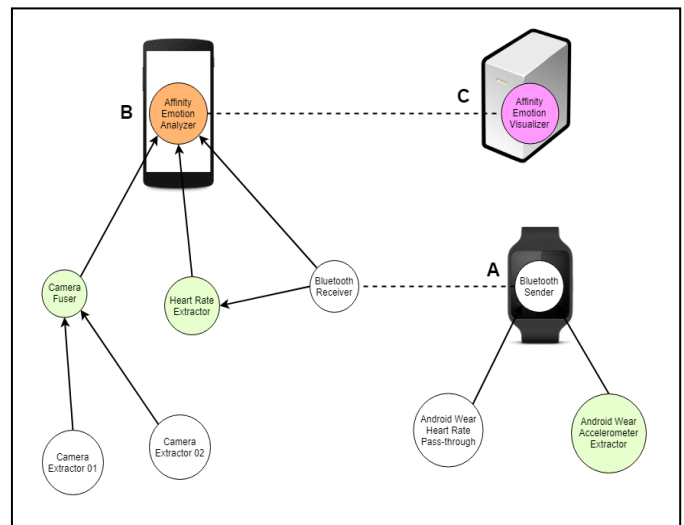


Figure 1. Informal diagram of the main test scenario

The Affinity Platform can be seen as an additional layer on top of regular software systems that connects them together in a protocol-agnostic way. This layer being also language-agnostic and OS-agnostic (Operating System), brings into the discussion a new, higher level programming where applications are written

by simply interconnecting Components. Given a sufficiently large repository of Components, complete software systems could be built from already developed Components.

The main objective of this architecture consists of the realization of chained data systems that can be easily distributed and offloaded among various devices, such as mobile phones, wearables, or autonomous vehicles. This system is meant to serve as an enabler for data science fields but is not limited to them, as this technology has the potential to substantially reduce development time and greatly improve collaboration between vastly different fields.

II. RELATED WORK

Most plugin architectures rely on strict application programming interfaces (APIs) [2] and give little customization options. Other systems, as for example Emacs [4], can be fully customized - including their internals, but have the disadvantage of requiring developers to learn a new and often cumbersome scripting language. Eclipse [5] exemplifies another category of plugin architecture - in Eclipse everything is a plugin, every item included in its menus and submenus. Every plugin has a manifest file that describes the way the plugin works and what interface it requires. Eclipse-like plugins seem to strike the best balance so far, but they are still too hard to use for regular users. React Components [6] are also greatly appreciated by the developer community. They are similar in functionality to plugins but have the advantage of being able to work independently of their parent system. The Affinity Platform builds on these predecessors but adds the concept of treating even fully-fledged applications as possible Components. This, together with its high level of generality, makes Affinity stand out among traditional plugin architectures and gives it the status of a platform for plugins. A comparison of these traits is presented in Table I.

Chimera [7] was chosen as the best fitting solution for linking Components due to its closely related philosophy to the Affinity Platform. It is a language-agnostic component-based framework capable of orchestrating independent applications. It is boasting similar ease of use and compatibility to the UNIX pipe system [8] while being more advanced, adding support for features like distributed and parallel computing.

TABLE I. COMPARISON OF THE AFFINITY PLATFORM FROM THE POINT OF VIEW OF PLUGIN ARCHITECTURES

	Plugin Architectures			
	<i>Emacs</i> [4]	<i>Eclipse</i> [5]	<i>React</i> [6]	<i>Affinity</i>
Extendable	✓	✓	✓	✓
Full customizability of resulted systems	✓	X	✓	✓
Full control over the granularity of plugins	✓	✓	✓	✓
Independent plugins	X	X	✓	✓
Easy to use for individuals	X	X	✓	✓
No previous programming knowledge required	X	X	X	✓
High level platform	X	X	X	✓

Chimera was considered as the starting point of the Affinity Platform but needed a series of improvements. The main complaint with Chimera is that it requires users to learn the CHIML [9] markup language. To improve on this, a graphical user interface (GUI) was designed in ordered to empower users to easily discover Components and connect them together. It makes use of diagram representations to express software systems in a way that can be easily managed and understood. Main features of this GUI include the ability to search for the desired Component, add a Component to the diagram, change the configuration options of a Component, and connect the channels of two Components. In the end, a diagram can be used to automatically generate the corresponding CHIML script. This interpretation phase allows for certain optimization procedures to be carried out, such as seamlessly adding parallelization to the resulting script.

In order to chain Components running on mobile devices, the terminal emulator Termux was used to achieve compatibility with the Node.js run-time environment on the Android operating system. Making use of Termux, CHIML scripts were able to run on an Android mobile phone. The main differences between the mentioned piping mechanisms are illustrated in Table II.

III. OVERVIEW

The Affinity Platform is built on the concept of fully independent Components. They are considered the building blocks of Affinity-developed software systems but are fully capable of operating completely on their own outside of their parent system. This kind of architecture closely follows core software engineering principles such as being low in coupling and high in cohesion.

Compatibility is a strong point as any existing application can be easily integrated into the Platform as long as it can be invoked by using the command line or is actively listening for data. Each Component must define a number of input channels and output channel, together with a set of configuration options that can be specified by the user. These characteristics can be defined using JavaScript Object Notation (JSON) in an “affinity_component.json” file or, for applications built using Node.js or related frameworks, they can be included straight into the “package.json” file.

TABLE II. COMPARISON OF THE AFFINITY PLATFORM FROM THE POINT OF VIEW OF SOFTWARE PIPING MECHANISMS

	Piping Mechanisms		
	<i>UNIX Pipe</i> [8]	<i>Chimera</i> [7]	<i>Affinity</i>
Language agnostic	✓	✓	✓
Distributed computing	X	✓	✓
Parallel computing	X	✓	✓
Support for mobile platforms	X	X	✓
No previous programming knowledge required	✓	X	✓
Visual interface	X	X	✓

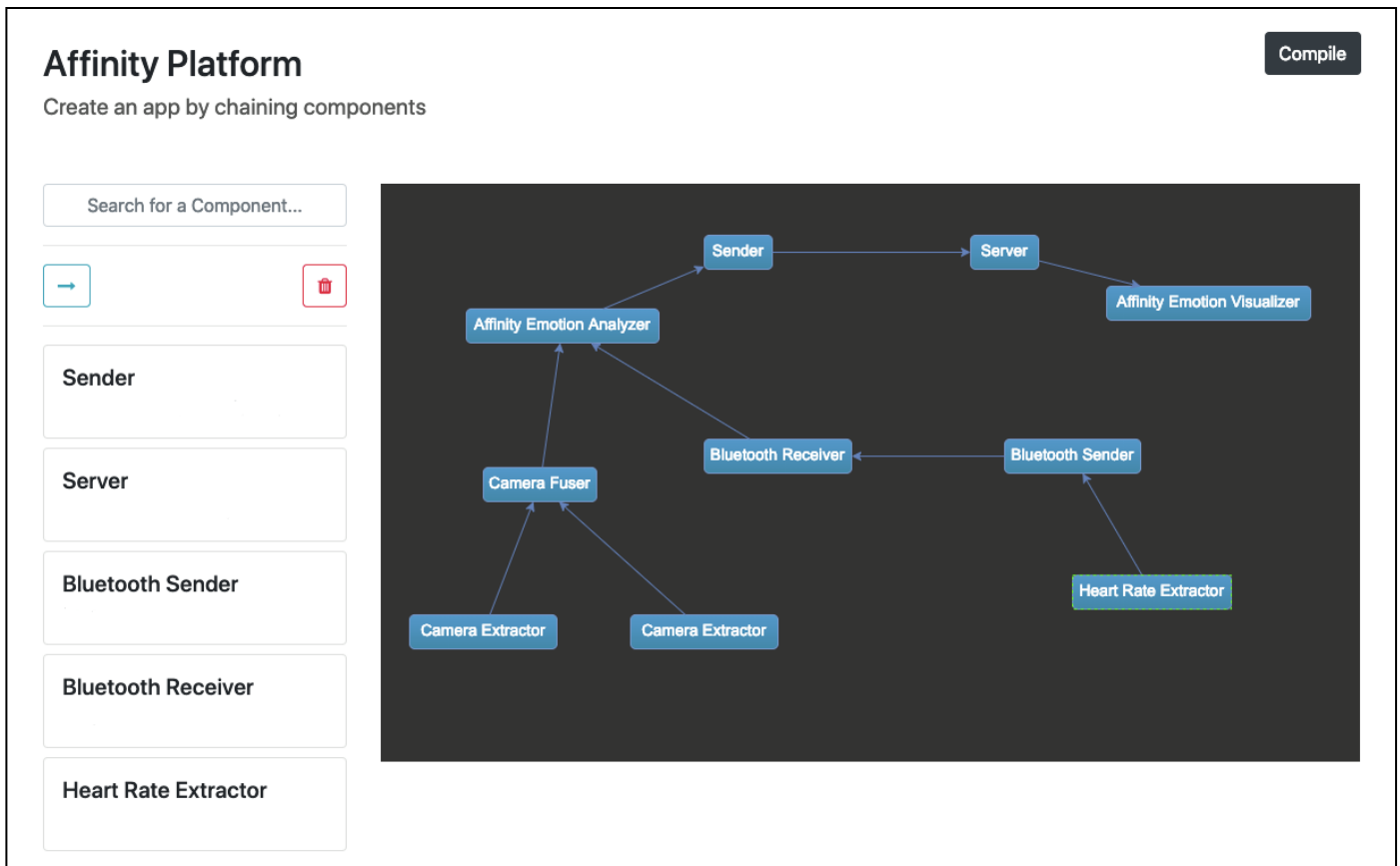


Figure 2. The graphical user interface of the Affinity Platform

Other than building software systems from scratch, the Affinity Platform can also be used to add plugins support to already existing applications. This is done by simply exposing input and/or output channels of the app, transforming it into a Component. Afterward, new modules can be added by connecting other Components to these channels.

Connections between Components are specified from an output channel to an input channel and can be routed in varying degrees of complexity. Interpolating Components gives developers abilities like broadcasting data to multiple Components, selectively sending data to the best available machine, and fusing together data coming from multiple Components before it gets processed.

The way inputs and outputs are connected is described in a CHIML configuration file. To make writing these files easier, a GUI application for generating CHIML configurations out of system diagrams is made available to developers. Diagrams are built using the mxGraph [10] JavaScript library, making advanced editing options readily available. The current state of the visual interface employed by the Affinity Platform can be observed in Figure 2. At the moment, finishing touches are added to analyzing these diagrams and generating optimized CHIML files out of them. Scripts auto-generated from diagrams can be edited in case a lower level of control is desired.

Components can be connected through the Chimera piping system or through any communication channels such as

Hypertext Transfer Protocol, WebSocket, Remote Procedure Call, Bluetooth Low Energy etc. implemented by other Components. The inter-compatibility of input and output channels is not guaranteed as Components are not limited by the communication medium or data representation. These are left as choices and they are not imposed as a responsibility of the Component. Developers can easily achieve compatibility between dissonant Components by defining inputs and outputs that support different data representations or by incorporating additional Components to act as translators between common data representations or communication mediums.

IV. RESULTS

The Affinity Platform was envisioned during the development of the main Affinity software application. A simpler way of making the Affinity Emotion Analyzer and its related algorithms available to the public was needed. A high level of reliability and ease of use were desired in order for other researchers to not just be able to extend and reuse this work but to actually make it easy to do so.

Using the Affinity Platform, the main Components of the Affinity application are exposed in an individual manner. This means, for example, that if somebody wants to use just the Affinity Emotion Analyzer, it could easily take it out of the system and integrate it.

The results confirm that using the Affinity Platform can improve the reusability of a software system and its Components up to a state where they can be considered future-proof. In this regard, the life expectancy of a Component is similar to that of an audio plugin. A Component becomes obsolete only when there is no longer a machine that can run its code or in the improbable case that its communication medium is not supported anymore and there is no way of adapting it to a different medium. This is different from the way in which regular plugins are added to an application as it doesn't require strict version compatibility. In extreme cases, a component could be run on a totally different system, in a Docker container for example, where all its dependencies are already met and thus ensuring long-lasting compatibility.

What is even more impressive is that this is achieved with minimal overhead on the part of developers. The only change that they need to make is adding a simple manifest file to their apps in order for the Affinity Platform to be able to identify them as Components and manage their input and output channels. This manifest file contains only the critical information about how to invoke a Component, how to transfer data through it (more specifically, its channels), and what configuration options are available for it. Usually, this file is only a few lines long. An example of how a minimal manifest file looks like is given in Figure 3.

V. DISCUSSION

The results obtained in this experiment are indicative of the potential of the Affinity Platform. Nonetheless, these are only early results and what could be accomplished goes far beyond what was presented in this experiment.

At this moment, the Affinity Platform is built around the concept of Components but, as better understanding is gained over the capabilities of these Components, we start to see how smart and independent they could become in the future. Components with the ability to reshape the connections of the system that they are part of are an interesting proposition. They could be the basis for integrating intelligent agents [11] in the Affinity Platform, using them to create smart connections between Components - connections made as a result of the data the agent has about its other Components. For now, the platform is closer related to component-based systems but it has strong ties to intelligent agents and might prove to be a powerful tool for building agent-oriented systems [12].

Concrete challenges of such a platform could be better understood when an affective aware developer aims for high precision in multimodal mobile affective applications [13].

```
{
  "in": ["Camera Extractor 1", "Camera Extractor 2"],
  "out": ["Fused features"],
  "command": "camera-fuser -2"
}
```

Figure 3. Example of an affinity_component.json file

Real life (mobile) facial expressions recognition systems are also demanding for high computing resources [14].

VI. CONCLUSION AND FUTURE WORK

This paper discusses the practical and theoretical aspects of using a data-driven plugin architecture in Affective Computing [3] and, more specific, context-aware applications.

Another area where this architecture could lead to substantial advancements is in Artificial Intelligence (AI) systems for writing software programs. As Components become more specialized, they start to represent more granular pieces of programming code, chunks of code similar to the ones that programmers write intuitively after they get accustomed to solving a recurring problem. AI could use these pieces to write complete programs on its own by simply making connections between Components. By gaining knowledge about the Components used in a system, AI could also describe the use of a system and make educated guesses about what it can do.

Currently, intelligent agents are thought to be the next step in developing the Affinity Platform. Integrating Components that can alter the inner linking of a system in order to better adapt to their goal brings forward new ways of thinking about the platform.

ACKNOWLEDGMENT

This work was supported by MHP Romania under MPH Lab Cluj-Napoca – Babeş-Bolyai University collaboration project.

REFERENCES

- [1] A. Kaur, K. S. Mann, "Component based software engineering," *IJCA*, vol. 2, no. 1, pp. 105-108, 2010.
- [2] M. Piccioni, C. A. Furia, and B. Meyer, "An empirical study of API usability," in *Proc. ESEM*, Baltimore, MD, USA, 2013, pp. 5-14.
- [3] R. W. Picard, "Affective Computing," MIT Press, 1997.
- [4] Emacs, <https://www.gnu.org/software/emacs/> (Accessed: 15/10/2018).
- [5] The Eclipse plug-in architecture, https://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html (Accessed: 15/10/2018).
- [6] React components and props, <https://reactjs.org/docs/components-and-props.html> (Accessed: 30/12/2018).
- [7] G. F. Gunawan, M. Amien, J. F. Palandi, "Chimera - simple language agnostic framework for stand alone and distributed computing," *CAIPT*, 2017.
- [8] W. R. Stevens, *UNIX Network Programming*, Prentice Hall, Inc., 1990.
- [9] CHIML markup language, <https://github.com/goFrendiAsgard/chimera-framework/wiki/CHIML> (Accessed: 30/12/2018).
- [10] mxGraph version 3.9.12, <https://jgraph.github.io/mxgraph/> (Accessed: 01/02/2019).
- [11] M. Wooldrige, "Agent-based software engineering," Mitsubishi Electric Digital Library Group, 1997.
- [12] H. Yu, Z.Q. Shen, C.Y. Miao, "Intelligent software agent design tool using goal net methodology," *IAT* 2007.
- [13] K.I. Bența, M. Cremene, M.F. Vaida, "A multimodal affective monitoring tool for mobile learning," *RoEduNet NER*, Craiova, p. 34-38, 2015.
- [14] K.I. Bența, M.F. Vaida, "Towards real-life facial expression recognition systems," *AECE*, 15(2), pp. 93-102, 2015.

A Mapping Study about Data Lakes: An Improved Definition and Possible Architectures

Julia Couto, Olimar Borges, Duncan Ruiz, Sabrina Marczak, and Rafael Prikladnicki
School of Technology, PUCRS - Pontifical Catholic University of Rio Grande do Sul - Porto Alegre, Brazil
{julia.couto, olimar.borges}@edu.pucrs.br, {duncan.ruiz, sabrina.marczak, rafaelp}@pucrs.br

Abstract—In the past few years, data lakes emerged as a trending topic in big data technologies. Although literature presents different points of view related to its functionalities, it serves mainly to store a variety of data in a big data context. In this paper, we aim to identify and analyze data lake definitions and possible architectures. Our methodology was composed of a systematic literature mapping based on PRISMA, software engineering best practices to perform reviews, and Kappa method to assess results' quality. We performed the search in eight different electronic databases to achieve a wide variety of publishers in Computer Science. We first identified 662 papers matching our search criteria; after filtering, we selected 87 papers for review. We found that the term data lakes was first defined by James Dixon in 2010. We also found that the term is often related to raw data repositories. From the identified definitions, we propose a new one as a means to better state what data lakes refer to and improve how the community use them. Moreover, we found that Hadoop and its ecosystem compose the most used toolset to create data lakes, revealing that this is the mainstream in architectures for data lakes as of today's available technologies.

Index Terms—Data lakes, Big Data, Literature review, PRISMA, Hadoop.

I. INTRODUCTION

Data lakes are a recent and trending topic in big data context [24], [81]. It is often referred as an architecture to store big data. They are often compared to traditional data warehouses, but both concepts differ in several aspects. For instance, unlike data warehouses, data lakes can easily scale and have the ability to store schema-less and multivariate data that will be processed just when information needs to be extracted from the stored dataset [8], [13], [38], [44], [80]. This native characteristic makes data lakes quite suitable for big data ecosystems.

Although the concept was first used in earlier 2010 [20], it was adopted by academia only a couple of years later. Thus, there is no consolidated and universally accepted definition, and its functionalities vary according to the context. For example, some say it is only a data repository [30], [43], [86], while others say it is a complete ecosystem, from data acquisition to information visualization [9], [55], [62]. By having these different functionalities, data lakes also present different possible architecture configurations.

The foundation architecture to create a data lake may be different according to some variables, such as its purpose, the skills of the people responsible for creating it, the available

infrastructure, and tools. For example, when considering the Hadoop Ecosystem, there are more than a hundred tools available for it [22], with thousands of possible combinations among their use. Knowing which ones are most commonly used or those that are most commonly reported in the literature may be useful for novices on big data, who need to set up an initial data lake, and also for those who are more experienced users, who might want to know new tools to add advanced features to an existing data lake.

Therefore, the aim of our literature review was to better understand what definitions have been used by the research community for the term 'data lake', and to propose a more comprehensive definition to facilitate and improve its use. Additionally, we also aimed to identify which big data architectures are used to build a data lake as well as to map the associated tools to do so.

More specifically, we performed a systematic mapping study in eight electronic databases. From the 662 identified papers, we selected 87 papers for review after filtering. We used the PRISMA checklist [59] to help us improve the quality report of our study, and the process suggested by Kitchenhan et al. [11] to plan the steps to be followed. To enhance results quality and measure the level of agreement between the researchers, we used the Kappa [54] method. To reduce bias, two researchers analyzed the selected papers and two others were consulted to resolve disagreements.

Our study revealed that James Dixon was the first author to use the concept of data lake to refer to a solution to store raw data in a Hadoop ecosystem, in 2010 [20]. The first conference paper to cite the term is from 2014, by O'Leary [63]. We also found that the terms most frequently associated to data lake are: *store*, *raw*, *repository*, *formats*, *analysis*, *storage*, *processed*, and *sources*. About data lake architectures, Hadoop is the most commonly used, stand-alone or in combination with other tools, such as Spark and NoSQL databases. The remaining sections present our study and results in details.

II. MATERIALS AND METHODS

A systematic mapping study, also known as mapping study (MS), is a type of literature review, a research method largely used to understand the state of art of some subject, and it allows us to map its origins and also how it developed over time, based on research questions. To develop our MS, we follow the process defined by Brereton et al [11]. These authors suggest three phases, namely Plan, Conduct, and Document

TABLE I
PICO AND PICO DEFINITIONS

PICO	PICO
Population: Big data systems	Population: Big data systems
Intervention: Data lakes	Interest: Definitions and architectures
Comparison: Definition of data lakes	Context: Data lakes
Outcome: Definition of data lakes and big data architectures in data lakes ecosystems	

the review, having ten stages to develop these phases. We also use the Preferred Reporting Items for Systematic Review and Meta-Analysis Protocols (PRISMA-P) [59] checklist, that has a set of items that must be addressed to report a systematic review, as described next.

A. Plan Review

In the Planning Phase, we defined research questions, and developed and assessed the review protocol. This phase must be done carefully because of its basis all subsequent research.

1) *Specify Research Question:* Our main objective is to answer the following Research Question (RQ): ***What are the definitions and possible big data architectures in data lake ecosystems?*** To better explore the papers, we splitted RQ into two, so each accepted paper can answer one or two questions: 1) *What are the most common definitions to the term data lake?* 2) *Which system architectures are reported to be used in data lake ecosystems?*

Aiming to limit and clarify our scope, we followed the PICO (Population, Intervention, Comparison, and Outcome) and PICO (Population, Interest, and Context) methods. These were initially developed by Sacket [72], to facilitate the elaboration of research definitions. PICO are most used for quantitative studies, while qualitative studies usually apply PICO [89]. As a MS can contain both qualitative and quantitative data, we used PICO and PICO to help us elaborate our research question. We present the scope of our research in Table I.

2) *Develop Review Protocol:* We developed and applied our search protocol using digital libraries available in the internet. We defined control studies so we could validate our search strings. A control study is an primary study resulting from systematized research, and which is known to answer our research questions. We used it to check if the search strings are adequate: if the control papers are not returned during string adjustments, the strings need to be adjusted until they do so. We used the two papers listed in Table II as control papers. Table III lists the used electronic databases and search strings.

TABLE II
CONTROL PAPERS.

Control Study 1	Terrizzano, Ignacio G., et al. "Data Wrangling: The Challenging Journey from the Wild to the Lake." CIDR. 2015. [87]
Control Study 2	Madera, Cedrine, and Anne Laurent. "The next information architecture evolution: the data lake wave." Proc. Int'l Conf. on Management of Digital EcoSystems. ACM, 2016. [50]

TABLE III
SEARCH STRINGS FOR EACH ELECTRONIC DATABASES.

Database	Search String
Springer	search?query="data+lake"&facet-language="En"&date-facet-mode=between&showAll=true&facet-discipline="Computer+Science"
Google Scholar	allintitle: "data lake"
Scopus	TITLE-ABS-KEY (data lake) AND (LIMIT-TO (SUBJAREA , "COMP")
Web of Science	(from all databases): TOPIC: ("data lake") OR TITLE: ("data lake") OR AUTHOR IDENTIFIER: ("data lake")
IEEE Xplore	((("Document Title": "data lake") OR "Abstract": "data lake") OR "Author Keywords": "data lake")
Science Direct	Title, abstract, keywords: "data lake"
arXiv	order: -announced_date_first; page_size: 50; primary_classification: cs; terms: AND all="data lake"
ACM	acmdlTitle:(+data +lake) AND recordAbstract:(+data +lake) AND keywords.author.keyword:(+data +lake)

It is important to note that we did not set a data range for the search. Returned results from as early as 1969 to 2013 referring to data lakes, upon inspection, were identified to discuss geological lakes. Thus, given that these do not relate to Computer Science, we discarded them; having the first paper of interest reported in 2014.

3) *Validate Review Protocol:* Two researchers developed the review protocol, who made several trials changing the search string to obtain results relevant and aligned to the research question. Then, the protocol was validated by two other senior researchers with a PhD degree in Computer Science. One of these researchers is a domain specialist in databases and big data and the other in research methodology. The study was conducted based on the updated protocol upon their reviews, as presented next.

B. Conduct Review

We conducted the study as per the defined protocol.

1) *Identify Relevant Research:* We applied the defined search string and, from the results, generated a bibtex file format for each electronic database. Bibtext is a plain-text file-format that contains lists of references, with information about all paper that matches our search criteria.

2) *Select Primary Studies:* To reduce bias, we splitted the papers to be analyzed between two researchers. We start selection phase with 1st researcher reviewing and marking each paper as accepted or rejected. Then, we perform three review rounds, based on Kappa method [54], each one containing a random sample of 5% of the papers population that was reviewed by the second researcher. We used the Kappa statistic [54] to measure the level of agreement between the researchers. Kappa result is based on the number of answers with the same result for both observers [46]. Its maximum value is 1, when the researchers have almost perfect agreement, and it tends to zero or less when there are no agreement between them.

TABLE IV
KAPPA RESULTS, BASED ON LANDIS & KOCH [46].

Kappa values	Agreement	1° round	2° round	3° round
<0	Poor			
0 – 0,20	Slight			
0,21 – 0,40	Fair			
0,41 – 0,60	Moderate	0.42		
0,61 – 0,80	Substantial		0.64	
0,81 – 1	Almost perfect			0.82

For each round, the 2nd researcher received a sample, analyzed each paper, and marked each one as accepted or rejected. Then, we compared the answers: if 1st and 2nd researchers accepted the same paper, we have an agreement in that paper. Then, we calculate Kappa value for the round. After that, in the papers where there is no agreement, the two main researchers discuss about the paper to reach a consensus. If there is still no consensus, the other researchers are contacted to help decide.

Landis & Koch [46] define a scale to interpret the Kappa results (see Table IV). We can also see in this table the results from the 3 rounds of analysis. We can see that the level of agreement increased, from moderate in the 1st round to substantial in the 2nd one, and in the last we achieved almost perfect agreement. In each iteration, we discussed the results and the reasons why some papers had been selected and other had not, improving the agreement between the researchers on the next iteration. From that, the second researcher received the second-half of papers to independently review.

3) *Assess Study Quality*: In order to retrieve interesting results related to the research topic, we defined inclusion and exclusion criteria for the papers. To be accepted, papers must meet all the following criteria: 1) Be a qualitative or quantitative research on data lakes in data management; 2) Present a complete study in electronic format; 3) Be a conference paper, review or journal. On the other hand, papers we rejected meet at least one of the following criteria: 1) Incomplete or short paper (less than 3 pages); 2) Unavailable for download; 3) Not about data lakes in data management; 4) Duplicated study; 5) Written in another language than English; 6) Conference proceedings index.

4) *Extract Required Data*: To help us organize and classify the papers, we used a tool named StArt (State of the Art through Systematic Review)¹. StArt was developed by the Federal University of São Carlos, Brazil, and it helps researchers in the process of systematic literature reviews. StArt has a execution phase with 3 processes: studies identification, selection, and extraction. We first register the protocol, and then we register each database and import its bib file, then use StArt to help keep record of selected papers. After finishing all data extraction using StArt, we exported the results to a Google Sheets, so we could analyze the data.

5) *Synthesize Data*: We used Google Sheets to help analyze and summarize our results. It also helped to work collabora-

TABLE V
PAPERS PER ELECTRONIC DATABASES.

Source	Initial	Accepted
Scopus	108	53 papers: [1]–[3], [5], [9], [10], [13]–[19], [23]–[29], [31]–[33], [37], [40], [45], [49], [50], [57], [60]–[66], [68], [70], [71], [73], [76]–[78], [81]–[84], [88], [90], [91], [93]–[95]
Springer	222	20 papers: [4], [6], [12], [21], [30], [36], [38], [39], [41]–[43], [47], [51], [53], [69], [74], [79], [85], [86], [92]
Google Scholar	197	6 papers: [8], [34], [56], [67], [80], [87]
Web of Science	71	4 papers: [7], [44], [48], [58]
Science Direct	19	2 papers: [35], [75]
IEEE Xplore	32	1 paper: [52]
arXiv	7	1 paper: [55]
ACM	6	0 papers

tively, as Google Sheets is available online.

C. Document Review

1) *Write Review Report*: After finishing answering the questions of the MS, we use our protocol as a basis to document the Review. Results are presented in Section III.

2) *Validate Report*: Once we finished the report, it was independently reviewed by three senior researchers. Each one read and suggest improvements that were adjusted to this final version.

III. RESULTS

We started with 662 papers retrieved from the initial search through the web engines. During the process, we identified that 155 are duplicated, and 419 were rejected according to exclusion criteria previously explained. At the end of MS process, we accepted 87 papers, published between 2014 and 2018. Table V presents the distribution of papers per database. In this table, we can see that most of the papers came from Springer and Google Scholar. It happens because Springer does not allow us to refine the filter of the studies, so results contains lots of books and books chapters, which we reject, as we explained in inclusion and exclusion criteria. Google Scholar, in the same way, does not allow complementary filters, frequently redirects to other engines, and it also brings a lot of websites and non-scientific reports among the results.

Among the rejected papers, 75 were published before 2010, when the term data lake was first used in big data context. We have to manually reject the older ones due to the fact that in most electronic databases we cannot filter results to show only Computer Science related studies. The papers previous to 2010 are mostly from Geology or Civil Engineering. We found that there is an increasing interest in data lakes, since 2014, with most papers being published in 2018.

Other interesting aspect we can see in Table V is that more than half papers we accepted are from Scopus. It happens due to the fact that Scopus is the largest database of abstracts and scientific citations, compiling more than 71 million records, 23 million titles and 5,000 publishers, among them ACM, Elsevier, IEEE, Springer, etc. So, we probably accepted papers

¹ Available at: http://lapes.dc.ufscar.br/tools/start_tool

from other databases using Scopus reference, and then it was marked as duplicated in the original database version.

From the 87 papers we accepted, 71 present data lake definitions. We read each one and copied the definition they present to the term data lake. Then, we created a unique text containing all definitions, and we passed through a web tool to count the words. This tool removed the stopwords, the most frequent terms in English, that are usually removed before natural language processing. Then, it returned a list containing all the other words and the amount of times they appear in the text. We analyzed the resulting list and grouped the variances in the same word, by the most frequent one: e.g.: *analyses*, *analyzing*, and *analysis* were grouped into *analysis*. Table VI presents the top 30 most frequent words. Then, based on the most frequent words, we create a word cloud (Figure 1) and a new definition to the term data lake, presented below.

During our analysis, we mapped who the authors of the papers references when using a definition for data lakes. We found that James Dixon was the first one to use the term lake in big data context, in a post in its blog in 2010 [20], and he is referenced by ten papers [4], [6], [17], [32], [38], [44], [62], [63], [67], [91]. The first author to reference Dixon’s Concept in academic context was O’Leary [63], in a paper published in 2014. We also discovered the most cited academic definition for data lakes is from Terrizzano et al. [87], mentioned in twelve papers [5], [26], [27], [33], [34], [52], [66], [78], [82], [84], [91], [93].

TABLE VI
30 MOST FREQUENT WORDS RELATED TO DATA LAKE DEFINITIONS.

B. Which system architectures are reported to be used in data lakes ecosystems?

1) *Ingestion*: Class of tools that work on data acquisition and collection, from the most varied sources. In this group, the most cited tool is Apache Kafka, which consists of a high-capacity, low latency distributed streaming platform for real-time data processing.

3) *Processing*: Tools in this group are responsible for analyzing, processing and transforming the raw data, so we can extract information from it. In this group, Apache Spark is the most cited in all papers, besides Apache Hadoop. It is a framework for distributed computing that provides an interface for clustered programming with parallelism and fault tolerance.

TABLE VII
ARCHITECTURES: THE MOST USED TOOLS IN DATA LAKES

Tool	Amount	Papers
1) Ingestion		
Apache Kafka	10	[1], [6], [9], [31], [47], [55], [68], [76], [82], [85]
Apache Flume	7	[1], [6], [27], [52], [61], [70], [83]
Apache Sqoop	5	[1], [27], [47], [52], [55]
Apache Nifi	3	[1], [55], [76]
Komadu	2	[83], [84]
Talend Studio	2	[1], [93]
2) Storage		
Apache Cassandra	6	[1], [6], [21], [40], [41], [45]
MongoDB	6	[16], [33], [35], [41], [43], [62]
Apache HBase	4	[1], [31], [47], [69]
MySQL	4	[1], [33], [43], [84]
Neo4J	3	[65], [85], [91]
Oracle	3	[1], [12], [42]
Apache Mahout	2	[1], [42]
GlusterFS	2	[48], [64]
PostgreSQL	2	[41], [78]
3) Processing		
Apache Spark	26	[1], [6], [8], [9], [12], [18], [24], [27], [33], [35], [40], [42], [44], [47], [49], [51], [52], [55], [61], [68], [69], [71], [82]–[84], [92]
Apache Hive	11	[1], [6], [9], [12], [25], [27], [31], [43], [55], [61], [69]
Apache Storm	7	[1], [42], [55], [65], [82]–[84]
Apache Impala	4	[12], [61], [69], [88]
Apache Drill	4	[12], [43], [65], [71]
Apache Oozie	4	[1], [27], [52], [55]
Python	4	[1], [16], [35], [88]
Apache Flink	3	[6], [44], [82]
Apache Pig	3	[1], [27], [52]
Apache POI	2	[1], [66]
Kepler	2	[83], [84]
Shiny	2	[27], [52]
Splunk	2	[1], [69]
WEKA	2	[42], [93]
4) Presentantion		
Microsoft Power BI	2	[88], [90]
Tableau	2	[1], [61]
5) Security		
Apache Ranger	4	[31], [55], [70], [71]
Kerberos	3	[55], [70], [71]
Apache Ambari	2	[55], [71]
Apache Knox	2	[31], [71]
Apache Sentry	2	[31], [71]

5) *Security*: Includes tools to manage system authentication and authorization, assure data security, permit auditing, and allow data encryption. Apache Ranger is the most mentioned. It is a framework for activating, monitoring and managing data security in the Apache Hadoop ecosystem. According to our analysis, the Apache Software Foundation (ASF) develops most of the tools reported in the studies for data lake architectures, helping creating the most used ecosystems.

IV. CONCLUSION

In this paper, we presented a systematic mapping study to better explain data lakes definition and architecture. We started with 662 papers, and we end up with 87 in the final set, after our criteria selection. The papers we selected are from 2014 to 2018, and came from eight different electronic databases.

We learned that the term data lake was first used in 2010 to designate a big data system. We proposed a new definition from the selected papers in our study for the concept data lake. We also found that Hadoop and its ecosystem comprises the most frequent architecture to built data lakes.

One limitation of our study is that we choose to limit the search only to the papers that have the term "data lake". We know that many researchers can be working with data lakes without using this buzzword, but as we want to know its definition, we chose to accept that limitation. For future work, we plan to further the investigation on the used tools and architectures, discussing the categories we listed and building a framework to help beginners to choose the best configuration according to its needs.

REFERENCES

- [1] F. Ahmad *et al.*, "Qos lake: Challenges, design and technologies," in *SigTelCom*, 2017, pp. 65–70.
- [2] A. Ahmadov *et al.*, "Towards a hybrid imputation approach using web tables," in *BDC*. IEEE/ACM, 2015, pp. 21–30.
- [3] H. Alili *et al.*, "Quality based data integration for enriching user data sources in service lakes," in *ICWS*. IEEE, 2018, pp. 163–170.
- [4] H. Alrehamy *et al.*, "Semlinker: automating big data integration for casual users," *Journal of Big Data*, vol. 5, no. 1, p. 14, 2018.
- [5] A. Alserafi *et al.*, "Towards information profiling: Data lake content metadata management," in *ICDMW*. IEEE, 2016, pp. 178–185.
- [6] S. Auer *et al.*, "The bigdataeurope platform – supporting the variety dimension of big data," in *ICWE*. Springer, 2017, pp. 41–59.
- [7] A. Beheshti *et al.*, "Coredb: A data lake service," in *CIKM*. ACM, 2017, pp. 2451–2454.
- [8] R. Benaissa *et al.*, "Clustering approach for data lake based on medoid's ranking strategy," in *CSA*. Springer, 2018, pp. 250–260.
- [9] M. Bhandarkar, "Adbench: A complete benchmark for modern data pipelines," in *TPCTC*. Springer, 2017, pp. 107–120.
- [10] W. Brackenbury *et al.*, "Draining the data swamp: A similarity-based approach," in *HILDA*. ACM, 2018, pp. 13:1–13:7.
- [11] P. Brereton *et al.*, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of systems and software*, vol. 80, no. 4, pp. 571–583, 2007.
- [12] P. Ceravolo *et al.*, "Big data semantics," *Journal on Data Semantics*, vol. 7, no. 2, pp. 65–85, 2018.
- [13] B. Cha *et al.*, "International network performance and security testing based on dist. abyss storage cluster and draft of data lake framework," *Security and Communication Networks*, vol. 2018, pp. 1–14, 2018.
- [14] H. Chen *et al.*, "An early functional and performance experiment of the marfs hybrid storage ecosystem," in *IC2E*. IEEE, 2017, pp. 59–66.
- [15] Y. Chen *et al.*, "Enhancing the data privacy for public data lakes," in *ICASI*. IEEE, 2018, pp. 1065–1068.
- [16] A. Ciociola *et al.*, "Umap: Urban mobility analysis platform to harvest car sharing data," in *SmartWorld*. IEEE, 2017, pp. 1–8.
- [17] N. Dessì *et al.*, "Increasing open government data transparency with spatial dimension," in *WETICE*. IEEE, 2016, pp. 247–249.
- [18] A. Dholakia *et al.*, "Designing a high performance cluster for large-scale sql-on-hadoop analytics," in *Big Data*. IEEE, 2017, pp. 1701–1703.
- [19] C. Diamantini *et al.*, "An approach to extracting thematic views from highly heterogeneous sources of a data lake," in *SEBD*, 2018, pp. 1–12.
- [20] J. Dixon, "Pentaho, hadoop, and data lakes," <https://jamesdixon.wordpress.com/2010/10/14/>, 2010, accessed: 2019-02-20.
- [21] H. Dutta, "Graph based data governance model for real time data ingestion," *CSI Trans. on ICT*, vol. 3, no. 2, pp. 119–125, 2015.
- [22] J. R. *et al.*, "The hadoop ecosystem table," <https://hadoopecosystemtable.github.io/>, 2019, accessed: 2019-02-22.
- [23] H. Fang, "Managing data lakes in big data era," in *CYBER*. IEEE, 2015, pp. 820–824.
- [24] M. Farid *et al.*, "Clams: Bringing quality to data lakes," in *ICMD*. ACM, 2016, pp. 2089–2092.
- [25] A. Farrugia *et al.*, "Towards social network analytics for understanding and managing enterprise data lakes," in *ASONAM*. IEEE/ACM, 2016, pp. 1213–1220.

- [26] Y. Gao *et al.*, "Navigating the data lake with datamaran," *CoRR*, vol. abs/1708.08905, 2017.
- [27] I. García *et al.*, "Towards a scalable architecture for flight data management," in *DATA, INSTICC*. SciTePress, 2017, pp. 263–268.
- [28] S. Gollapudi, "Aggregating financial services data without assumptions," in *ICSC*. IEEE, 2015, pp. 312–315.
- [29] N. Golov *et al.*, "Big data normalization for massively parallel proc. databases," *Comp. Standards & Interfaces*, vol. 54, pp. 86 – 93, 2017.
- [30] C. Gröger, "Building an industry 4.0 analytics platform," *Datenbank-Spektrum*, vol. 18, no. 1, pp. 5–14, 2018.
- [31] M. Gupta *et al.*, "An attribute-based access control model for secure big data proc. in hadoop ecosystem," in *ABAC*. ACM, 2018, pp. 13–24.
- [32] R. Hai *et al.*, "Constance: An intelligent data lake system," in *ICMD*. ACM, 2016, pp. 2097–2100.
- [33] —, "Query rewriting for heterogeneous data lakes," in *ADBIS*. Springer, 2018, pp. 35–49.
- [34] A. Y. Halevy *et al.*, "Managing google's data lake: an overview of the goods system," *IEEE Data Eng. Bull.*, vol. 39, no. 3, pp. 5–14, 2016.
- [35] J. Herman *et al.*, "Using big data for insights into sustainable energy consumption in industrial and mining sectors," *Journal of Cleaner Production*, vol. 197, pp. 1352 – 1364, 2018.
- [36] J. Hui *et al.*, "Integration of big data: A survey," in *Data Science*. Springer, 2018, pp. 101–121.
- [37] M. Jarke, "Data spaces: Combining goal-driven and data-driven approaches in community decision and negotiation support," in *GDN*. Springer, 2017, pp. 3–14.
- [38] M. Jarke *et al.*, *On Warehouses, Lakes, and Spaces*. Springer, 2017, ch. 16, pp. 231–245.
- [39] P. Jovanovic *et al.*, *A Unified View of Data-Intensive Flows in Business Intelligence Systems: A Survey*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, ch. 3, pp. 66–107.
- [40] M. Karpathiotakis *et al.*, "No data left behind: Real-time insights from a complex data ecosystem," in *SoCC*. ACM, 2017, pp. 108–120.
- [41] N. Kasrin *et al.*, "Semantic data management for experimental manufacturing tech," *Datenbank-Spektrum*, vol. 18, no. 1, pp. 27–37, 2018.
- [42] L. Kassner *et al.*, "The stuttgart it architecture for manufacturing," in *ICEIS*. Springer, 2017, pp. 53–80.
- [43] P. Kathiravelu *et al.*, "A dynamic dw platform for creating and accessing biomedical data lakes," in *DMAH*. Springer, 2017, pp. 101–120.
- [44] P. P. Khine *et al.*, "Data lake: a new ideology in big data era," *ITM Web Conf.*, vol. 17, p. 11, 2018.
- [45] H. Kondylakis *et al.*, "Implementing a data management infrastructure for big healthcare data," in *BHI*. IEEE, 2018, pp. 361–364.
- [46] J. R. Landis *et al.*, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.
- [47] T.-H.-Y. Le *et al.*, "Big data driven architecture for medical knowledge management systems in intracranial hemorrhage diagnosis," in *IUKM*. Springer, 2018, pp. 214–225.
- [48] C. Li *et al.*, "The design and application of astronomy data lake in china-vo," in *ADASS*, vol. 512. ASP, 2017, p. 157.
- [49] A. Maccioni *et al.*, "Crossing the finish line faster when paddling the data lake with kayak," *VLDB*, vol. 10, no. 12, pp. 1853–1856, 2017.
- [50] C. Madera *et al.*, "The next information architecture evolution: The data lake wave," in *MEDES*. ACM, 2016, pp. 174–180.
- [51] K. P. Maksymowicz *et al.*, "A holistic approach to testing biomedical hypotheses and analysis of biomedical data," in *BDAS*. Springer, 2016, pp. 449–462.
- [52] M. A. Martínez-Prieto *et al.*, "Integrating flight-related information into a (big) data lake," in *DASC*. IEEE/AIAA, 2017, pp. 1–10.
- [53] S. McCarthy *et al.*, "Combining web and enterprise data for lightweight data mart construction," in *DEXA*. Springer, 2018, pp. 138–146.
- [54] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.
- [55] J. McPadden *et al.*, "A scalable data science platform for healthcare and precision medicine research," *CoRR*, vol. abs/1808.04849, 2018.
- [56] S. D. Meena *et al.*, "Data lakes - a new repository for big data analytics," *Int. Journal of Adv. Research in CS*, vol. 7, no. 5, pp. 65–67, 2016.
- [57] N. Miloslavskaya *et al.*, "Big data, fast data and data lake concepts," *Procedia Computer Science*, vol. 88, pp. 300 – 305, 2016.
- [58] S. Mitrovic, "Specifics of the integration of business intelligence and big data technologies in the processes of economic analysis," *Business Informatics*, vol. 42, no. 4, pp. 40–46, 2017.
- [59] D. Moher *et al.*, "Preferred reporting items for systematic review and meta-analysis protocols," *Systematic reviews*, vol. 4, no. 1, p. 1, 2015.
- [60] B. M. Mrozek *et al.*, "Soft and declarative fishing of information in big data lake," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 5, pp. 2732–2747, 2018.
- [61] A. A. Munshi *et al.*, "Data lake lambda architecture for smart grids big data analytics," *IEEE Access*, vol. 6, pp. 40 463–40 471, 2018.
- [62] I. D. Nogueira *et al.*, "Modeling data lake metadata with a data vault," *CoRR*, vol. abs/1807.04035, 2018.
- [63] D. E. O'Leary, "Embedding ai and crowdsourcing in the big data lake," *Intelligent Systems*, vol. 29, no. 5, pp. 70–73, 2014.
- [64] E. Pena *et al.*, "Framework to use modern big data software tools to improve operations at the paranal observatory," in *Proc. SPIE*, vol. 10704, 2018, pp. 10704 – 10704 – 11.
- [65] A. Pomp *et al.*, "Enabling semantics in enterprises," in *ICEIS*. Springer, 2018, pp. 428–450.
- [66] C. Quix *et al.*, "GEMMS: A generic and extensible metadata management system for data lakes," in *CAISE*, 2016, pp. 129–136.
- [67] K. Rajesh *et al.*, "An introduction to data lake," *i-manager's Journal on Information Technology*, vol. 5, no. 2, pp. 1–4, 2016.
- [68] R. Ramakrishnan *et al.*, "Azure data lake store: A hyperscale distributed file service for big data analytics," in *ICMD*. ACM, 2017, pp. 51–63.
- [69] B. Ramesh, *Big Data Architecture*. New Delhi: Springer India, 2015, ch. 2, pp. 29–59.
- [70] S. Rangarajan *et al.*, "Scalable architecture for personalized healthcare service rec. using big data lake," in *ASSRI*. Springer, 2018, pp. 65–79.
- [71] P. Revathy *et al.*, "Analysis of big data security practices," in *iCATcT*, 2017, pp. 264–267.
- [72] D. L. Sackett, *Evidence-based Medicine How to practice and teach EBM*. WB Saunders Company, 1997.
- [73] H. B. Sankaranarayanan *et al.*, "Passenger reviews reference architecture using big data lakes," in *Confluence*, 2017, pp. 204–209.
- [74] R. S. Santos *et al.*, "Big data analytics in a public general hospital," in *MOD*. Springer, 2016, pp. 433–441.
- [75] S. Sharma, "Expanded cloud plumes hiding big data ecosystem," *Future Generation Computer Systems*, vol. 59, pp. 63 – 92, 2016.
- [76] G. Shlyuger, "Apply analytical grid processing to sensor data collections," in *SPIE*, vol. 10185, 2017, pp. 10 185 – 10 185 – 13.
- [77] K. Singh *et al.*, "Visual bayesian fusion to navigate a data lake," in *FUSION*, 2016, pp. 987–994.
- [78] T. J. Skluzacek *et al.*, "Klimatic: A virtual data lake for harvesting and distribution of geospatial data," in *PDSW-DISCS*, 2016, pp. 31–36.
- [79] G. V. Solar *et al.*, "Big data management: What to keep from the past to face future challenges?" *DS and Eng.*, vol. 2, no. 4, pp. 328–345, 2017.
- [80] R. K. Sreekala PK, "Data lake in the big data era: An overview," *Library Herald*, vol. 56, no. 1, pp. 11–15, 2018.
- [81] M. K. Srinivasan *et al.*, "State-of-the-art big data security taxonomies," in *ISEC*. ACM, 2018, pp. 16:1–16:7.
- [82] J. Stefanowski *et al.*, "Exploring complex and big data," *Int. J. Appl. Math. Comput. Sci.*, vol. 27, no. 4, pp. 669–679, 2017.
- [83] I. Suriarachchi *et al.*, "Crossing analytics systems: A case for integrated provenance in data lakes," in *e-Science*. IEEE, 2016, pp. 349–354.
- [84] —, "Provenance as essential infrastructure for data lakes," in *IPAW*, 2016, pp. 178–182.
- [85] Y. Taher *et al.*, "A service-based system for sentiment analysis and vis. of twitter data in realtime," in *ICSOC*. Springer, 2017, pp. 199–202.
- [86] —, "A context-aware analytics for processing tweets and analysing sentiment in realtime," in *OTM*. Springer, 2016, pp. 910–917.
- [87] I. G. Terrizzano *et al.*, "Data wrangling: The challenging journey from the wild to the lake," in *CIDR*, 2015, pp. 1–9.
- [88] S. Tovernic *et al.*, "Solution for detecting sensitive data inside a data lake," in *MIPRO*, 2018, pp. 1284–1288.
- [89] M. University, "Systematic reviews: Using pico or pico," <https://goo.gl/fqPoCY>, 2018, accessed: 2018-12-20.
- [90] W. Villegas-Ch *et al.*, "Big data, the next step in the evolution of educational data analysis," in *ICITS 2018*. Springer, 2018, pp. 138–147.
- [91] C. Walker *et al.*, "Personal data lake with data gravity pull," in *BDCloud*. IEEE, 2015, pp. 160–167.
- [92] R. Wenning *et al.*, *Compliance Using Metadata*. Springer, 2018, ch. 3, pp. 31–45.
- [93] M. Wibowo *et al.*, "Machine learning in data lake for combining data silos," in *DMBD*. Springer, 2017, pp. 294–306.
- [94] S. Yadav *et al.*, "Business data fusion," in *Fusion*, 2015, pp. 1876–1885.
- [95] T. Yamada *et al.*, "Interactive service for visualizing data assoc. using a self-org. structure of schemas," in *SOCA*. IEEE, 2017, pp. 230–233.

Architecture for Discovery and Customization of Multi-tenant Learning Process as a service and resources allocation in cloud computing

Sameh Azouzi
ISITCom Hammam Sousse
Laboratory RIADI-GDL, ENSI,
Mannouba, Tunisia
Azouzi_sameh@yahoo.fr

Sonia Ayachi Ghannouchi
ISG Sousse/ Laboratory RIADI-GDL,
ENSI, Mannouba, Tunisia
s.ayachi@coselearn.org

Zaki Brahmi
ISITCom Hammam Sousse
Laboratory RIADI-GDL, ENSI,
Mannouba, Tunisia
zakibrahmi@gmail.com

Abstract— Collaborative e-learning based on several means of electronic communication mechanism specifically internet and web2.0 (such as forum, virtual classroom, videoconference, etc.) has become the current trend in e-learning systems. The cloud computing environment presents itself an important infrastructure for supporting these collaborative e-learning systems and for optimizing these service-based systems in a multi-tenant way. In this paper, we discuss the problem of discovering multi-tenant e-learning processes. For this, we propose an architecture for a configurable learning process discovery environment and the allocation of resources (Saas, Paas, Iaas) necessary for the execution of the corresponding learning activities.

Keywords- *Discovery e-learning process; collaboratif e-learning; business process; resource allocation; cloud computing.*

I. INTRODUCTION

E-learning is defined as Internet-enabled learning [1]. E-Learning components can include multi-format content, knowledge sharing, inclusion of collaboration tools, and social networking. It is important to emphasize the value of collaboration and knowledge sharing between learners and teachers in e-learning systems. In this context, great attention is given by researchers, developers and educators to collaborative learning through Learning Management Systems (LMS). However, several studies in literature show some limitations of these e-learning platforms (LMS) [4] [5] [6]. These limitations are mainly at the level of collaboration and interaction between different actors of an e-learning activity; they based mainly on texts and photos. The form and effect of collaboration and interaction are relatively simple and limited.

DOI reference number: 10.18293/SEKE2019-136

So, it is unrealistic for each school to offer its own e-learning system that responds to learner expectations and manages the new neo-digital society. In order to remedy these different LMS deficiencies, we have contributed in previous work [7] to a general, collaborative, configurable and multi-tenant e-learning system. The collaboration we propose is synchronous and asynchronous. And we also suggest using the business process line approach (BPL) inspired by the Software Product Line (SPL) approach for the development of multi-tenant e-learning applications. The proposed general e-learning process could be implemented in a variety of contexts and accommodates the different needs of universities and teachers. However, all variants of our model share a set of common elements / activities and others differ by some variable activities.

Thus, the adoption of an SPL and BPL approach in the field of e-learning seems to be a promising solution. On the one hand, it overcomes the limitations of LMS systems, and on the other hand, it provides institutions with e-learning applications tailored to their needs.

Motivated by the difference and diversity of the needs of tenants (teachers / learners), institutions are looking for e-learning systems available outside their organizations that are rapidly adapting to the new demands of their teachers and learners and reducing costs of development and maintenance. Cloud computing is the best solution because of its ability to outsource service-based learning processes with a pay-per-use model.

According to the National Institute of Standards and Technology (NSIT), cloud computing is a model that allows providers to share their resources (services, networks, storage servers, and applications) and the users who access them in a ubiquitous, and convenient way of request [8]. In a multi-tenant environment, the use of a configurable learning process provides configurable learning that can be configured by different tenants based on their specific needs

[9]. Different approaches for modeling multi-tenant configurable processes have been proposed; mainly they focused on configurable business process discovery and focused on configurable resource allocation. Thus, in this sense, we propose a discovery architecture of learning services considered as business processes in the cloud. In our architecture, we also offer discovery of other SaaS, PaaS, IaaS cloud services and collaborative web2.0 services that facilitate the deployment and execution of all types of learning activities in different contexts. But some activities require a wide bandwidth or a very powerful CPU; other activities require collaborative web2.0 applications and SaaS services to be executed.

In order to overview all these aspects, this contribution is arranged as follows.

In Section 2 we introduce the main concepts of Cloud Computing, e-learning including its infrastructure and main layers, configurable learning process, multi-tenant and resource allocation.

Next, Section 3 describes our proposed architecture. Finally, the main concluding remarks are given in Section 4.

II. PRELIMINAIRES

In this section, we give a brief description of related topics, which are utilized in this work.

A. Configurable learning process

The world of education and training has embraced new methods and new pedagogical tools. Since the computer, at first, and the Internet, then, have appears in classrooms, departments but also and especially in the daily life of each of the actors involved, the school (in the most generic sense of the term) is engaged in a real race to stay the more possible in line with the reality of society. Each school faces a major challenge, which is the development of a learning system that manages IT innovations and meets the needs of teachers / learners knowing that they do not cease to evolve. Starting from this point, there comes the idea of developing a configurable e-learning process. A configurable process model allows the sharing of a process between different tenants, which can be customized according to specific needs. To model a configurable learning process, one must study the variability of this process. In our work, we view the e-learning process as a multi-tenant business process and each tenant can choose a sub-process (configuration) of the overall process. With respect to business process variability, tenants can have varying business workflows. Therefore, the application must allow configuration and customization to meet the tenant's goals and requirements [23]. In previous work we have presented the modeling of a multi-tenant configurable general e-learning process [7,24].

B. Cloud computing and e-learning

Cloud computing uses a new technology that develops applications in a way that allows both the execution of the application and the storage of data to be performed ubiquitously for all users. Cloud architecture provides user support at different levels that vary according to their

different characteristics. The different levels can be described as follows [3]:

- Infrastructure as a Service (IaaS): remote management and control of hardware resources provided by a system.
- Platform as a Service (PaaS): offers the cloud platform along with a series of libraries to develop applications in which the distribution of tasks, the persistence and other layers are transparent for the developer.
- Software as a Service (SaaS): consists in offering different applications to be used through the internet as opposed to a local installation.

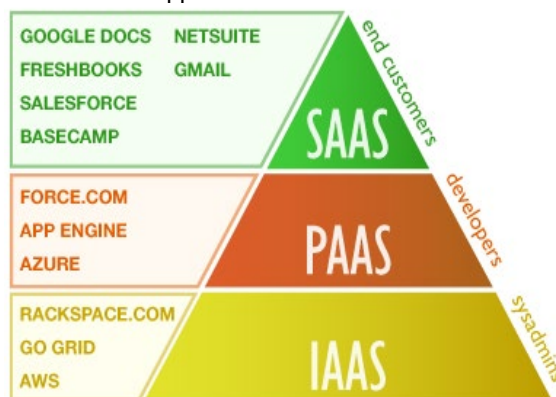


Figure 1. Illustration of the layers for the Services Oriented Architecture [12]

There are incipient developments that broach the topic of cloud technology and e-learning [2], however few studies incorporate both concepts. The success of the solution software as a service (SaaS) is real and is easily applicable for e-learning. Cloud computing, SaaS and e-learning are completely complementary. Accordingly, we note that the focus on changing traditional e-learning systems to more collaborative interactive learning environments has increased. J. Méndez & J. Gonzalez [3] highlights limitations of traditional e-learning pointing that system creation and maintenance are placed inside the educational institutions, which can cause a lot of problems including huge investments with no gain and lack of development potential. In 2009, cloud computing is presented by [20] as a new computing model to implement e-learning ecosystem to overcome the problems in the traditional system. The author considers that cloud computing is able to add some critical features to e-learning ecosystem, such as: configuration at real-time, utilization of resources, on demand resource sharing and better management for software or hardware [20]. The cloud-based system supports the construction of a new generation of e-learning systems that is accessible from a wide range of hardware devices, while storing data inside the cloud.

B. Dong & al., [22] proposes a platform architecture based on the integration of cloud computing and web 2.0 for developing intelligent virtual learning community and makes the learning environment more productive, scalable, flexible and adjustable towards students demands and needed

information and communication technologies. The author states that the usage of cloud computing and web2.0 for e-learning affects the way e-learning software projects are managed and the proposed intelligent virtual learning community enhances the efficiency of learning environment, provides up-to-date resources, constancy, guaranteed quality of service, dependability, scalability, minimized time, efficient usage of resources, flexibility, and maintaining of e-learning system.

Within the same context, Ouf, Nasr and Helmy [21] have proposed an e-learning system based on the integration of cloud computing and Web 2.0 technologies to meet the requirements for e-learning environment such as flexibility and compliance towards students' needs and concerns, improve and enhance the efficiency of learning environment.

Aljenaa & al., in [23] introduced the main components of a system of effective e-learning through using cloud computing technology. The authors insist on the importance of the following components:

- "Cloud software platform" which contains the LMS platform and the necessary tools of collaboration and communication to meet the needs of learners throughout the process of learning;
- "Operational and management components" to make the management of the learning process;

C. Multi-tenancy

Multi-tenancy being one of the key features of service in cloud, the service providers can offer single application to multiple users. A multi-tenant application caters a personalized customer experience improving performance and efficiency and provides a flexible space for customers to efficiently pursue new business demands. From the software vendor perspective, such multi-tenant applications, improve resource utilization and reduce the operational costs in delivering the software as a service. Each of the tenants may have unique requirements that differ from each other. Customization plays a vital role to converge specific requirements of every tenant. It is up to the vendor, what extent of customization to be offered to the tenant [10]. Two main goals of multi-tenancy are isolation and sharing, which are contradictory [11]. As we discussed in the introduction, the primary motivation of multi-tenancy is to enable management of the needs of universities/learners that are rapidly changing and diversified. However, at the same time, each user needs complete isolation and often does not want to know even the existence of the other users. Furthermore, a multi-tenant setup will inevitably attract more requests, and therefore must be scalable. Scalability in a multi-tenant setup has several dimensions: that is supporting large number of requests, processes, and tenants [11].

D. Resource allocation in cloud-based Business process

In previous work, we considered e-learning processes as business processes, because of the similarity between the two in the orchestration of learning activities and that of business

processes [7]. Thus, we use the BPFM approach for modeling configurable e-learning processes in the cloud. Running a learning process requires IaaS-level resources that can be distributed across virtual machines (VMs), SaaS services and applications, and PaaS platforms. Until today, research on business process discovery in the cloud and resource allocation has been rare. There are works such as [13, 14], which focus on the allocation of human resources. Some other works explicitly consider the characteristics of the cloud: In [18], the authors proposed an approach for configurable cloud resources allocation in multi-tenant business process. Their aim is to shift the cloud resource allocation from the tenant side to the cloud process provider side for a centralized resource allocation management. Through configuration, different tenants can easily customize the selection of the needed resources taking into account two important properties: elasticity and shareability.

S. Schulte & al. In [15, 16] develop a Platform for Elastic Processes, which combines the functionalities of a BPMS with that of a Cloud resource management system. Recently, work in the area of configurable process modeling has been proposed with perspectives for resource allocation. La Rosa & al. In [25] Offer C-EPC (configurable event driven process chains) with resource, data and physical object capture capabilities. However, they focus on human resources and there is no support for cloud resources. In [17], A. Hallerbach & al. Extend process variants with options (Provop) to properly model and manage large collections of process variants. Juhnke & al. [18] provide an extension to a standard BPEL workflow engine, which allows the use of Cloud resources to execute business processes. The same applies to the work by Bessai & al. [19], who also assumes that workflows are composed of single software services. The authors offer different methods to optimize resource allocation and scheduling, pareto-optimal solution covering both cost and time. E. Hachicha & al. [26] proposed an approach for configurable cloud resource allocation in multi-tenant business processes and to shift the cloud resource allocation from the tenant side to the cloud process provider side for a centralized resource allocation management. Table I summarizes these approaches and relates them to properties that are important in a cloud setting. We observe that resource variability, cloud features, and allocation are only partially covered or not at all. In the following, we aim to fill these gaps by the definition of our novel approach which allows the configurable e-learning process discovery and allows the cloud resources allocation needed for the learning activities execution such as IaaS, PaaS and SaaS.

TABLE I. Evaluation of previous approaches

Criteria Approach	Control- flow variability	Multy- tenancy	Ressource variability	Ressource allocation Saas	Ressource allocation Iaas	Ressource allocation Paas	BP- discovery in cloud
[18]	-	+	-	-	+	-	-
[15, 16]	+	-	-	-	+	-	-
[17]	+	+	-	-	-	-	-
[19]	-	-	-	-	+	-	-
[26]	+	+	+	-	+	-	-
Our approach	+	+	+	+	+	+	+

E. Discussion

We can conclude by considering that all these research works concerning the systems of e-learning in the cloud, integrating the multi-tenant aspect and allocating the resources needed to execute an e-learning process suffer from some limits namely:

The absence of a complete initiative or approach for the construction of an agile and reconfigurable process of learning adaptable to change and being able to evolve to meet the needs of learners;

- Lack of collaboration in learning processes and limited re-use possibilities from the processes deployed by other universities/teachers;
- The allocation of resources in the cloud is limited to the infrastructure level.

In our research work, we considered an e-learning process as a multi-tenant configurable business process in the cloud (BPaaS) and to achieve this goal we combined the Software Product Line (SPL) and Business Process Management approach (BPM).

In the next section we will:

- Provide an approach for discovering a multi-tenant configurable e-learning process and configurable cloud resource allocation.
- The allocation of cloud resources (IaaS, PaaS, SaaS) configurable in multi-tenant e-learning process.

III. MULTI-TENANCY AWARE CONFIGURABLE LEARNING PROCESS AS A SERVICE DISCOVERY ARCHITECTURE

The goal of this research work is the development of multi-tenant, configurable and collaborative learning process, that utilizes BPM and SPL for modeling the learning process and cloud computing. In cloud computing environments, there are two actors: the providers and the cloud computing users. On the one hand, suppliers hold massive resources in their large data centers and rent these resources to users. On the other hand, there are users who have applications with various loads and rent resources from the providers for running their applications. Once the final deployment solution has been defined and the learning process is deployed, multi-tenant configurable learning process description are created and published by learning process providers into service registries hosted in cloud data centers. In the datacenter registers, there are also descriptions of other cloud services namely SaaS, PaaS, IaaS and others collaborative Web2.0 services that have been used in the execution of the process. Published learning process as a services artifact are specified as business processes. The proposed discovery architecture provides personalized learning process configurations for tenants as presented in figure 2.

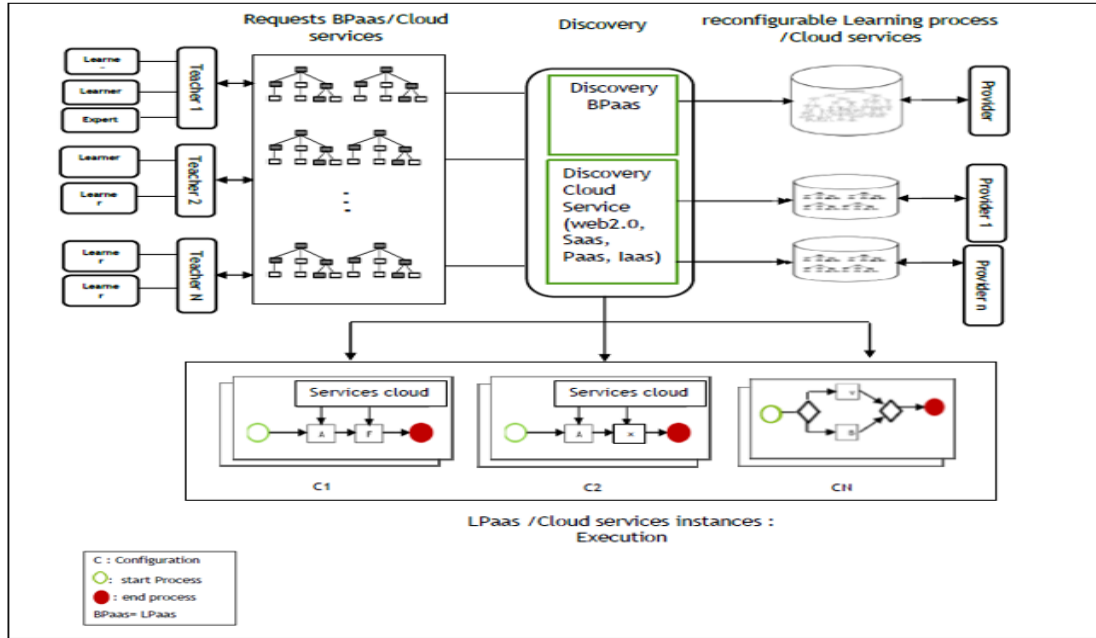


Figure 2. Conceptual architecture of Multi-tenant aware configurable learning process as a service discovery architecture and allocation of cloud resources

The discovery system consists of a modular architecture organized into separate modules: Request BPaaS/ Cloud services, Discovery, Reconfigurable learning process/ Cloud services, LPaaS/ Cloud services instances (execution).

Each module is composed of one or more components cited as follows:

- **Reconfigurable learning process / cloud services:** Our configurable learning process is created and published in configurable service registries hosted in cloud data centers. Thus, cloud services such as SaaS, PaaS, IaaS and collaborative web2.0 services are published by several cloud service providers. These services will be used in the execution of the e-learning process. This is what the allocation of resources required for the execution of such activity requires.
- **Requests BPaaS/ Cloud services:** Tenants who are in our case Institutions / Teachers look in the registers for configurable e-learning process configurations. They can formulate their required process configuration by selecting a set of activities. So, they look for cloud services that they want to use in running selected activities.
- **Discovery:** this module consists of two sub-modules: BPaaS discovery and cloud services discovery. The "BPaaS Discovery" module run to locate, discover and generate the appropriate configuration. This module uses another cloud service discovery module "discovery cloud service" to discover collaborative web2.0 services and other services (SaaS, PaaS, IaaS). These services will be used to carry out the selected activities, collaborate with experts, share knowledge and pedagogical resources, and strengthen the infrastructure to carry out such a task.

- **LPaaS / cloud services instances (runtime):** The return of the discovery module are instances of BPaaS / cloud services that respond to a request configuration. Once the recommended configuration is generated, it will be run as a business process in the cloud.

The multi-tenant aware discovery of configurable business process as a service is one of the most important and difficult issues, because of multiplicity and non-standardization of their description in cloud. In this paper, we introduce the combination of BPM approach and SPL approach for modeling collaborative, configurable e-learning process; then we introduce our architecture for discovery of the learning process in cloud. Our approach empowers multiple tenants to discover their desired learning process service configured variants, considering individual variants. To do so, we reduce the problem of configurable matching to a tree matching problem and we are going to adapt existing algorithms for this aim. We are doing the necessary experiments to show the feasibility of our approach.

IV. CONCLUSION

In this work, we exposed the limitations of existing e-learning systems (LMS) and discussed our contribution in previous work on modeling a collaborative, configurable and multi-tenant e-learning process. with the use of the Business Process Feature Model (BPFM) approach. Subsequently, we exposed the basic concepts of deploying a learning process as a business process (BPaaS) in the cloud: the different layers of cloud computing, the multi-tenant aspect and the supply of resources for running a process in the cloud.

For this, we propose an architecture of a discovery framework of a learning process and the allocation of resources cloud computing which allows the personalization

of e-learning services and the selection of collaborative web 2.0 services, services to infrastructure level and platform level.

In our future work, we are going to engage in showing the feasibility of our architecture where we rely on the structural matching approach and the graph-edit distance for e-learning BPaaS discovery in the cloud.

REFERENCES

- [1] Mayer, R., Clark, R.: E-Learning and the Science of Instruction: Proven Guidelines for Consumers and Designers of Multimedia Learning, 3rd edn. Pfeiffer (2011).
- [2] Ercan, T.: Effective use of cloud computing in educational institutions. *Procedia Social and Behavioral Sciences* 2, 938–942 (2010).
- [3] J. Méndez, J. Gonzalez, "Implementing motivational features in reactive blended learning : Application to an introductory control engineering course", *Education, IEEE transaction*, vol. 54, no. 4, pp. 619-627, 2011.
- [4] V. Stantchev, R. Colomo-Palacios, P. Soto-Acosta, and S. Misra, "Learning management systems and cloud file hosting services: A study on students' acceptance," *Computers in Human Behavior*, Vol. 31, February 2014, pp. 612–619.
- [5] M. A. Conde, F. García, M. J. Rodríguez-Conde, M. Alíer, and A. García-Holgado, "Perceived openness of Learning Management Systems by students and teachers in education and technology courses," *Computers in Human Behavior*, Vol. 31, February 2014, pp. 517-526.
- [6] Z. Du, X. Fu, C. Zhao, Q. Liu, and T. Liu, "Interactive and Collaborative E-Learning Platform with Integrated Social Software and Learning Management System," *Proceedings of the 2012 International Conference on Information Technology and Software Engineering*, Lecture Notes in Electrical Engineering, Vol. 212, 2013, pp 11-18.
- [7] Sameh Azouzi, Sonia Ayachi Ghannouchi, Zaki Brahmi: "Modeling of a Collaborative Learning Process with Business Process Model Notation", *International Conference on Digital Economy*, pp. 95—104, 2017.
- [8] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011).
- [9] Aalst, W.: Business Process Configuration in The Cloud: How to Support and Analyze Multi-Tenant Processes? In: *ECOWS*, IEEE (2011) 3-10.
- [10] Ankit Bhilwar, Sandesh Jain, others: "Multi-tenant enabled e-Learning platform: Blended with workflow technologies", *e-Learning, e-Management and e-Services (IC3e)*, 2014 IEEE Conference on, pp. 88—92, 2014.
- [11] Milinda Pathirage, Srinath Perera, Indika Kumara, Sanjiva Weerawarana: "A multi-tenant architecture for business process executions", *Web services (icws)*, 2011 IEEE international conference on, pp. 121—128, 2011.
- [12] Fernandez, A, Peralta, D, Herrera, F., et al. An overview of e-learning in cloud computing. In : *Workshop on Learning Technology for Education in Cloud (LTEC'12)*. Springer, Berlin, Heidelberg, 2012. p. 35-46.
- [13] Cabanillas, C., et al.: Towards process-aware cross-organizational human resource management. In: *BPMDS, EMMSAD CAiSE*, Greece, June 16-17. (2014) 79-93.
- [14] Kajan, E., et al.: The network-based business process. *IEEE IC* 18 (2014) 63-69.
- [15] Schulte, S., et al.: Costdriven optimization of cloud resource allocation for elastic processes. In: *International Journal of Cloud Computing*. (2013) 2326-7550.
- [16] Schulte, S., et al.: Realizing elastic processes with viep. (In: *Service-Oriented Computing - ICSOC*, China, Nov 12-15, 2012, Revised Selected Papers) 439-442.
- [17] Hallerbach, A., et al.: Capturing variability in business process models: The provop approach. *Journal of Software Maintenance and Evolution: R & P* (2010) 519-546.
- [18] Juhnke, E., Dörnemann, T., Bock, D., Freisleben, B. (2011). Multi-objective Scheduling of BPEL Workflows in Geographically Distributed Clouds, *Proc. of IEEE 4th Intern. Conf. on Cloud Computing (CLOUD 2011)*, Washington DC, USA, 412-419.
- [19] Bessai, K., Youcef, S., Oulamara, A., Godart, C. (2013). Bi-criteria Strategies for Business Processes Scheduling in Cloud Environments with Fairness Metrics, *Proc. of IEEE 7th Intern. Conf. on Research Challenges in Information Science (RCIS 2013)*, Paris, France, 1-10.
- [20] D. Chandran, "Hybrid E-learning platform based on cloud architecture model: A proposal", in *Signal and Image Processing (ICSIP)*, 2010 International Conference on, Chennai, 2010, pp. 534 - 537.
- [21] S. Ouf, M. Nasr, and Y. Helmy, "An enhanced e-learning ecosystem based on an integration between cloud computing and Web2.0?" in *Signal Processing and Information Technology (ISSPIT)*, 2010, pp. 48 - 55.
- [22] B. Dong, Q. Zheng, J. Yang, H. Li, M. Qiao, "An e-learning ecosystem based on cloud computing infrastructure", in *Advanced Learning Technologies*, 2009. *ICALT 2009*. Ninth IEEE International Conference, Riga, 2009, pp. 2009125 - 127.
- [23] Assylbek Jumagaliyev, Jonathan Nicholas David Whittle, Yehia Said Shahat Ahmed Elkhatib: "Evolving multi-tenant SaaS cloud applications using model-driven engineering", 2016.
- [24] Azouzi Sameh, Zaki Brahmi, and Sonia Ayachi Ghannouchi. "Customization of multi-tenant learning process as a service with Business Process Feature Model." *Procedia Computer Science* 126 (2018): 606-615.
- [25] Rosa, M.L., et al.: Configurable multi-perspective business process models. *Inf. Syst.* 36 (2011) 313–340
- [26] Hachicha, Emna, et al. "A configurable resource allocation for multi-tenant process development in the cloud." *International Conference on Advanced Information Systems Engineering*. Springer, Cham, 2016.

An Empirical Study about Software Architecture Configuration Practices with the Java Spring Framework

Quentin Perez, Alexandre Le Borgne, Christelle Urtado, and Sylvain Vauttier

LGI2P, IMT Mines Ales, Univ Montpellier, Ales, France

{Quentin.Perez, Alexandre.Le-Borgne, Christelle.Urtado, Sylvain.Vauttier}@mines-ales.fr

Abstract

Software architecture modeling plays a key role in software development and, beyond, in software quality. The Spring framework is widely used in industry to deploy software. This paper evaluates whether Spring fosters good practices for architecture definition. It describes the results of an empirical study, based on a corpus of open-source Spring projects. Analysis shows that a strong (70%) majority of projects mixes all Spring architecture definition features. This can be considered as a pragmatic use of a very flexible tool. However, few good practice documentation and tool assistance exist to prevent hazardous architecture constructions. The paper highlights these situations and concludes on recommendations to assist developers.

Keywords: Software architecture, architecture deployment, architecture configuration, empirical software engineering, Spring framework, GitHub open-source project.

1 Introduction

Architecture design is a critical issue that impacts software engineering [5]. Architectures are the natural consequence of modularity: They compose software from elementary components that can easily be developed, tested, maintained or reused.

Spring is a popular industrial framework designed for architecture development in Java. As established by a survey involving 2044 developers¹, it is the most used framework for web-service development. Spring has evolved over time with technologies (*e.g.*, adoption of Java annotations) or ap-

plication needs (*e.g.*, automation of deployment). Spring now provides developers with multiple features, that complement one another and sometimes overlap, for the architecture definition.

The remainder of this paper is organized as follows. Section 2 motivates this work. Section 3 first describes the features provided by the Spring framework then compares them with respect to expected qualities. Section 4 describes the empirical study on developers' practices. First, the methodology used for GitHub data extraction is presented. Then, the statistical analysis is developed before identifying threats to the study's validity and presenting the implemented tool that supports our study. To conclude, Section 5 presents related works and Section 6 draws perspectives for this work.

2 Motivations

Architecture models are used in nearly all steps of the software lifecycle, from its early design, as an abstract, ideal solution to meet users requirements, to its actual deployment and execution. Academic research has proposed many architecture description languages (ADLs) to support architecture conceptual design [11] whereas industry has proposed frameworks for runtime architecture deployment and management [4]. The ideal technology should both be flexible and easy to use for software developers and help document the architecture, increase component reusability and maintainability and manage change for software architects.

Defining a software architecture amounts to describe its components and their connections (the links that support their interactions). In the case of Java software, this amounts to define the constituent objects and the reference bindings to be created at runtime. When no specific archi-

¹<https://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-2016/>

texture management feature is available, architecture construction is classically hard-coded in the main procedure. Otherwise, an architecture deployment descriptor can be defined through a framework to set the architecture up. To compare their modeling capabilities, we have established a set of expected qualities:

Explicitness. Explicit architecture models are defined with dedicated elements, clearly separated from source code.

Declarativity. Declarative architecture models are defined by abstract elements, that specify the expected instantiated structures, not the instantiation code.

Encapsulation. Encapsulated architecture model definitions are not scattered across source code but gathered into modules.

Assistance. Assisted architecture design is supported by tools that verify consistency, use of good practices or architectural styles, and control evolution to prevent architecture drift or erosion [13].

The purpose of this study is to compare the different kinds of architecture descriptors provided by Spring and to run an empirical analysis to determine whether industrial practice is influenced by architecture model qualities.

3 Spring Features and their Qualities

Since Spring 4 (2013), three closely related architecture definition features are offered to developers:

XML descriptors. Architectures are defined by several XML descriptors that are parsed and interpreted at runtime by the Spring container. Architecture components are declared by the `<bean>` tag, which must define an identifier (*id*) and the instantiated class (*class*). Listing 1 defines a small home automation architecture composed of an *Orchestrator* object connected to a *Lamp* and a *Clock*. Connections are defined by binding bean properties to bean references, as declared by *property* tags. Bean properties correspond to component dependencies (actually instance attributes). These component dependencies are supplied by the Spring container using the declared beans in the XML descriptor (dependency injection).

Configuration classes. Architectures can alternatively be defined by specific Java classes, identified by the `@Configuration` annotation (see Listing 2). Configuration classes are automatically detected by the Spring container and executed to build the architecture. Beans are declared by methods annotated by `@Bean`. This enables to program all the necessary pre- or post- bean instantiation treatments required to manage complex settings. Connections are handled by passing bean references to bean constructors, as for the *myOrchestrator* bean in Listing 2, or to bean property setters. In Listing 2, the *clock1* and *lamp1* methods are thus used to retrieve the bean references passed to the *myOrchestrator* constructor. Being genuine Java, configuration classes

leverage IDE tools and compile-time verification, as type-safe bean connections and scoped identifiers.

Self-annotated classes. Architecture definition is integrated to the code of the supporting classes thanks to annotations. The `@Component` annotation identifies the classes that will be automatically instantiated by the container to create architecture beans (see Listings 3 and 4)². Similarly, the `@Autowired` annotation, which can be associated to attributes, setters or constructors, identifies dependencies (*i.e.*, connections) that will automatically be supplied by the container, using corresponding existing beans (retrieved by name or type). On the one hand, self-annotated classes are the most declarative way to define architectures. On the other hand, architecture definitions are scattered through and mixed with source code.

Moreover, Spring also supports any combination of the aforementioned architecture definition features.

Feature Quality Analysis. XML descriptors and configuration classes enable explicit and encapsulated architecture definitions. Regarding modularity, configuration classes leverage the object-orientation of Java. Considering these three qualities, configuration classes are the best choice and self-annotated classes the worst. This analysis is coherent with technical literature that recommends to limit the use of self-annotated classes to small projects [8, 14, 15]. Besides, self-annotated classes define architectures only with singleton classes. Rather than a limitation, this constraint is intended to enforce strong cohesion between class and architecture structures. Two antagonist approaches of architecture definition are thus supported: architectures that are orthogonal (generic and flexible) or integrated (to avoid architectural drifts) to the code of classes. Self-annotated classes therefore are recommended for projects that are subject to frequent changes [1, 2, 14]. When it comes to choosing features for architecture definition, technical literature only provides scarce guidance, often considering this choice as a matter of developers' tastes [16, 17, 15]. Furthermore, to our knowledge, the qualities of architecture definition mixing features have not yet been studied. We expect that these feature combinations result from rational decisions of experienced developers that use the most adapted feature to different parts of architectures.

4 Empirical Study of Developers' Practices when Using Spring Framework

Data Extraction. A corpus of 524 projects has been extracted from GitHub. In order to consider only significant, quality projects, we applied the selection criteria proposed

²Specialized annotations, like `@Service`, `@Repository` and `@Controller`, have been derived from `@Component` to identify the roles of beans in specific architecture kinds, like web-service architectures.

```
<beans
xmlns="http://www.springframework.org/schema/beans">
  <bean class="my.smartHome.Clock" id="clock1"/>
  <bean class="my.smartHome.Lamp" id="lamp1"/>
  <bean class="my.smartHome.Orchestrator" id="myOrchestrator">
    <property name="time" ref="clock1" />
    <property name="light" ref="lamp1" />
  </bean>
</beans>
```

Listing 1: XML descriptor bean configuration

```
@Configuration
public class BeansConfiguration{
  @Bean
  public Clock clock1(){return new Clock();}
  @Bean
  public Lamp lamp1(){return new AdjustableLamp();}
  @Bean
  public Orchestrator myOrchestrator()
  {return new Orchestrator(clock1(),lamp1());}
}
```

Listing 2: Annotation-based bean configuration

```
// Clock.java file extract
@Component
public class Clock implements Time {/...*/}
// Lamp.java file extract
@Component
public class Lamp implements Light {/...*/}
```

Listing 3: Annotated components

```
// Orchestrator.java file extract
@Component
public class Orchestrator{
  @Autowired
  private Time time;
  @Autowired
  private Light light;
}
```

Listing 4: Autowired configuration

in Jarczyk *et al.* [7], like the numbers of forks and stars. We thus extracted the last commit of projects with 100 stars or more, forked at least 10 times, written in Java, containing the “Spring” keyword and created after 2010-01-01 (*i.e.*, after Spring release 3). For reproducibility purposes, our metadata is available online³.

Empirical Analysis. To understand the state-of-practice, we first analyzed which Spring features were used in the studied corpus. Surprisingly, a majority of Spring projects mixes architecture definition features ($\simeq 69.3\%$) and, despite their qualities, configurations classes are only used in a minority of projects ($\simeq 6.5\%$). They are challenged by XML descriptors ($\simeq 12.6\%$) and self-annotated classes ($\simeq 11.6\%$). Apart from routine, in the case of XML descriptors, which are the oldest proposed feature, declarativity may thus be a key quality in developers’ decisions. Figure 1 presents the distribution of all the combination of architecture definition features, depending on the size of the projects measured with the Source Lines Of Code (SLOC) metric. As expected, self-annotated classes alone are only used in small projects. Surprise comes again from configuration classes, that are used alone in only rather small projects. Explicit and encapsulated architecture descriptions do not appear to be a primary concern. Again, declarative features are rather used in bigger projects and even the biggest ones. More interestingly, the biggest project seems to require the support of all the features together.

To confirm the intuition that project size has an impact on used features, we evaluate two hypotheses using a non-parametric statistical Kruskal-Wallis test:

Null hypothesis	Alternative hypothesis
\mathcal{H}_0 : No influence of architecture definition features on project size.	\mathcal{H}_1 : Influence of architecture definition features on project size.

³<https://github.com/DedalArmy/MISORTIMA/tree/data-study-spring-deploy-features>

Defining risk $\alpha = 5\%$, the result of the test is $H \simeq 93.68$ with a p -value of $\simeq 5.196^{-18}$. As p -value $\leq \alpha$, the null hypothesis \mathcal{H}_0 is rejected with a 5% risk. This demonstrates that architecture definition features have a significant influence on project size. As the choice of architecture definition features by the developer obviously does not determine the size of the project, we can infer that the correlation we measure is the reciprocal relation of the actual situation: the choice of architecture definition features is influenced by the size of the project. It would be interesting to study whether the choice of the technique is done *a priori* or evolves, depending on the size of the project.

Finally, we also analyzed isolatedly the use of self-annotated classes on the corpus of 524 Spring projects, depending of their size, using a chi-square test. This test rejects the hypothesis of a relation between project size and use of self-annotated classes, confirming the intuitive analysis of Figure 1. Usage of self-annotated classes thus seems definitely motivated by declarative convenience rather than sound modeling capabilities.

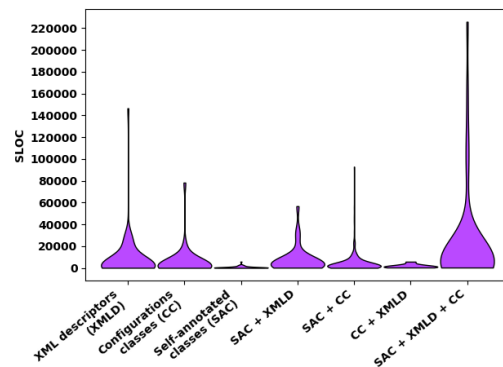


Figure 1: Distribution of architecture definition features regarding SLOC

Threats to Validity. One of the major issues regarding validity of our study is its generalization. This paper focuses exclusively on the Spring framework although Spring is only one among many architecture definition frameworks. This focus might bias the study. It would thus be interesting to explore developers' practices while they use other frameworks in order to compare results with those of this paper. Moreover, by using data provided and mined through the GitHub API we are confronted to the threats already identified by Kalliamvakou *et al.* [9].

5 Related Works

Several works on the deployment of component architecture have already been carried out. Parrish *et al.* [12] modeled the deployment of component architectures in a formal way. It makes it possible to describe several deployment strategies while ensuring deployment safety in terms of installation and component compatibility. Dearle [3] compared the different methods for architecture deployment and showed that dependency injection, as implemented in Spring, is a desirable mechanism for component lifecycle management. Another study has shown that Spring reduces the developers' workload, in particular by extra flexibility, as compared to Java EE [6]. To our knowledge, there is a lack of research on architecture definition combinations and different practices in terms of deployment configuration even if a strong technical literature exists on the subject [8, 14, 15, 16, 17].

6 Conclusion

The empirical analysis we have run on a corpus of 524 projects extracted from the hosting GitHub service about the usage of the architectural definition features provided by the Spring framework leaves opened questions. It shows that usage is strongly related to project size and thus results from rational developer decisions. However, usage seems to be motivated by rather practical than quality concerns, as shown by the predominant use of combined features including self-annotated classes in any size of projects. A first perspective is to study more precisely how Spring features are combined according to project size or domain. We also plan to compare features from other technologies (languages) and frameworks.

Previous work has shown that it is possible to recover an explicit architecture documentation by mining Spring XML configuration files and compiled source code [10]. However, the relevance of results depend on the quality of the architecture modeled in the code. The goal of the on-going work presented in this paper is complementary: fostering the best architectural qualities when source code is used as a standalone model in agile processes.

A more practical perspective is to pursue the development effort to try and better assist developers in their architecture deployment activities by visualization and development of assistance tools.

References

- [1] J. Carnell. *Spring Microservices in Action*. Manning, 2017.
- [2] I. Cosmina, R. Harrop, C. Schaefer, and C. Ho. *Pro Spring 5: An in-depth guide to the Spring framework and its tools*. Apress, 5th edition, 2017.
- [3] A. Dearle. Software deployment, past, present and future. In L. C. Briand and A. L. Wolf, editors, *FOSE workshop at 29th ICSE*, pages 269–284, Minneapolis, USA, May 2007. IEEE.
- [4] S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. *IEEE Transactions on Software Engineering*, 35(4):573–591, April 2009.
- [5] D. Garlan. Software architecture: A roadmap. In *FOSE track at 20th ICSE*, pages 91–101, Limerick, Ireland, June 2000. ACM.
- [6] P. Gupta and M. C. Govil. Spring Web MVC framework for rapid open source J2EE application development: a case study. *IJEST*, 2(6):1684–1689, 2010.
- [7] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki. GitHub projects. quality analysis of open-source software. In L. M. Aiello and D. McFarland, editors, *6th international SocInfo conference*, volume 8851 of *LNCS*, pages 80–94, Barcelona, Spain, Nov. 2014. Springer.
- [8] T. M. Jog. *Learning Spring 5.0*. Packt, 2017.
- [9] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining GitHub. In *11th Working MSR conference*, pages 92–101, Hyderabad, India, May 2014. ACM.
- [10] A. Le Borgne, D. Delahaye, M. Huchard, C. Urtado, and S. Vauttier. Recovering three-level architectures from the code of open-source java Spring projects. In X. He, editor, *30th international SEKE conference*, pages 199–202, San Francisco, USA, July 2018.
- [11] A. Mokni, C. Urtado, S. Vauttier, M. Huchard, and Z. Huaxi Yulin. A formal approach for managing component-based architecture evolution. *Science of Computer Programming*, 127:24–49, Oct. 2016.
- [12] A. Parrish, B. Dixon, and D. Cordes. A conceptual foundation for component-based software deployment. *JSS*, 57(3):193–200, July 2001.
- [13] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Software Engineering Notes*, 17(4):40–52, Oct. 1992.
- [14] D. Rajput. *Spring 5 design patterns: Master efficient application development with patterns such as proxy, singleton, the template method, and more*. Packt, 2017.
- [15] A. Sarin and J. Sharma. *Getting Started with Spring Framework: A Hands-On Guide to Begin Developing Applications Using Spring Framework*. CreateSpace, 3rd edition, 2016.
- [16] C. Walls. *Spring in action*. Manning, 4th edition, Nov. 2014.
- [17] N. S. Williams. *Professional Java for Web Applications*. Wiley & sons, 2014.

Recover and Optimize Software Architecture Based on Source Code and Directory Hierarchies

Tong Wang, Yelian Zhang, Xufang Gong, Bixin Li
School of Computer Science and Engineering
Southeast University, Nanjing, China

Abstract—Software architecture helps developers understand and maintain software, so how to obtain accurate architecture is critical. The architecture recovery technique is a widely used method for obtaining architecture. In order to improve the accuracy and efficiency of the architecture recovery technique, we propose a method for recovering and optimizing software architecture based on source code and directory hierarchies. Our method consists of three steps: first, we extract architecture-related information from source code and directory hierarchies and construct a file dependency graph; then, we preprocess and cluster code elements to construct a preliminary architecture; finally, we optimize architecture by clustering code elements based on the structural similarity and renaming components. We perform our method on four representative open source projects and compare our method with representative architecture recovery techniques. The experimental results show that the architectures recovered by our method has higher accuracy with higher efficiency than the compared architecture recovery techniques.

Index Terms—architecture recovery; clustering; architecture optimization

I. INTRODUCTION

Software architecture provides a high-level abstraction view to developers, so it is useful for understanding software to make evolution plans, reduce development costs, improve software performance [1] and improve software quality [2], [3], so how to obtain accurate software architecture is a hotspot issue.

The automated recovery technique mainly includes two phases, information extraction and information-based recovery. In the information extraction phase, the data sources commonly include source code and directory hierarchies [4]. The directory hierarchy is set by developers, so it reflects the logical relations between files [5]. These data sources provide valid information for software architecture recovery. However, most architecture recovery techniques usually only use one of them to recover architecture resulting in the low accuracy. In the information-based recovery phase, there are many types of recovery algorithms. However, there is a huge amount of code elements in large scale programs, so how to improve efficiency is also an important problem.

In summary, the current automated architecture recovery technique has the following problems. a) The data source is single. When information is missing, the accuracy would be reduced. b) In a large scale program, the number of code

elements is huge, so it will take a long time to recover architecture.

In order to solve the above problems, we propose the method for architecture recovery and optimization based on source code and directory hierarchies (ARO). The main contributions of this paper are mainly reflected in the following three aspects.

- The architecture-related information is extracted from source code and directory hierarchy to improve accuracy. The information of source code reflects the specific implementation of the architecture. The information of directory hierarchies is related to developers' design.
- The file dependency graph is preprocessed to improve efficiency. The file dependency graph is preprocessed to reduce the number of clustered code elements for reducing the time consumed, then we perform K-center-hierarchy algorithm to cluster code elements around the core code.
- The architecture is optimized. We optimize architecture by clustering based on the structural similarity and renaming components.

II. RELATED WORK

The automated recovery technique mainly includes two phases, information extraction, and information-based recovery.

Mainstream data sources contain the following types. a) Source code. Text analysis or feature localization of the source code of the program by constructing a platform similar to lexical analysis and grammar analysis, and the use of information retrieval technology to discover associations between documents. b) Documents. Documents related to software are collected, such as software design documents, UML diagrams, code comments, user manuals, etc., to establish the concept [6]. c) Directory hierarchy. Some methods consider the information to identify components, that is, it is a supplement to the information obtained by other methods and improves the accuracy of partitioning components [4].

Mainstream methods of information-based recovery contain the following types. a) Domain knowledge. Domain-based methods are performed in a top-down process or a bottom-up process. In the top-down process, the documents related to architecture materials are used to recover architecture. In the bottom-up process, according to the comments, the declaration of variable names, etc., using domain knowledge to recover

B. Li is the corresponding author. E-mail: bx.li@seu.edu.cn
DOI reference number:10.18293/SEKE2019-045

architecture [7]. b) Clustering. This type of method uses mathematical methods to study and process the classification [8]. c) Machine Learning. The information of code elements is trained to recover architecture. Machine learning-based methods are generally not used alone, but as a supplement to clustering algorithms to improve the accuracy of clustering. d) Pattern Matching. The recovery process is modeled as a mapping between the high-level abstraction and the code elements. It is a semi-automated technique that requires manual participation, and graph matching also requires a lot of computer resources and time [9].

According to the above analysis, we find that only the clustering algorithm and the machine learning method do not require additional manual intervention. However, it is difficult to obtain the training set [10]. Therefore, the clustering algorithm is more widely used in the automatic recovery architecture technique.

III. OUR METHOD

Our method contains the following steps: extracting information, preprocessing and clustering code elements, and optimizing architecture.

A. Extracting information

The information of source code is extracted by automatically analyzing source code, such as the dependency information between code elements and the basic information of code elements. The process of extracting information from source code contains two steps. a) Construct an abstract syntax tree. The source code is converted into an abstract syntax tree which is an intermediate representation. b) Analyze the abstract syntax tree. By analyzing the abstract syntax tree, the information between the code elements is extracted. A code element is a unit of code, such as a file, a package, and so on. The dependency between code elements can be divided into multiple types, such as the reference dependency between files, the generalization dependency between classes, and so on. Then we integrate dependency information to construct the file dependency graph.

The information of directory hierarchies reflects the logical dependencies between files, and it is mainly used in the following two aspects: a) Aggregating components. In source code, some files do not have dependencies with other files, such as test case files. These files are presented as an independent component in architecture. A large number of independent components have effects on the understandability, so we cluster independent files into an independent component. b) Adjusting components. The relations between files of the same directory do not strictly comply with the high cohesion principle and the low coupling principle, so we adjust components based on the dependencies between directories.

B. Preprocessing and clustering code elements

Before we perform the clustering algorithm, we preprocess the file dependency graph to reduce the number of files, resulting in reducing the time consumed.

In the preprocessing process, independent files are clustered as early as possible, which is beneficial to reduce the number of nodes in the dependency graph and to comply with the high cohesion principle and the low coupling principle. Then, we preprocess the file dependency graph by clustering code elements which are related to the types of strong dependency and the structure of strong dependency. The dependency type between code elements indicates the degree of the closeness between them. The common types of strong dependency between code elements contain the following types: the generalization dependency, the implementation dependency, the combination dependency, the definition dependency, and the declaration dependency. Five dependency structures belong to the structure of strong dependency. The structure of strong dependency contains the following structures: the single dependency, the tightly coupled, the closed-loop dependency, and the open-loop dependency.

After preprocessing the file dependency graph, we cluster code elements based on the distance between code elements. The distance is calculated based on the dependence intensity and the similarity of the directory.

Dependency Intensity(DI) represents the degree of the closeness between two code elements, and it depends on the dependency type and the number of dependencies. The DI between the code element x and the code element y is calculated as the following formula.

$$DI_{ab} = \frac{\sum_{i=1}^N \alpha_i Num(i, A, B)}{\ln(LOCA)} \quad (1)$$

where N represents the number of dependency types between a and b , i represents the i_{th} dependency type, $Num(i, a, b)$ represents the number of i dependency between a and b , α_i represents the weight of the dependency type.

Directory similarity(DirSim) describes the degree of the similarity between directories. If two code elements are in the same directory, the directory similarity is 1. The directory similarity between the code element a and the code element b is calculated as the following formula.

$$DirSim(a, b) = \frac{|Dir(a) \cap Dir(b)|}{|Dir(a) \cup Dir(b)|} \quad (2)$$

where $DirSim(a, b)$ is directory similarity between the code element a and the code element b , the $Dir(a)$ represents the directory path of a , $|Dir(a)|$ represents the directory depth of a .

Element distance(ET) is used to describe the distance between two code elements. We consider the distance between code elements based on the directory similarity and the dependency strength. The ET between the code element a and the code element b is calculated as the following formula.

$$ET(a, b) = DirSim(a, b) * DI_{ab} \quad (3)$$

where $ET(A, B)$ is the code element A and the code element B , $DirSim(A, B)$ represents the directory similarity between A and B , and DI_{AB} represents the dependence intensity between A and B .

The clustering efficiency is low when it deals with large amounts of code elements. Therefore, ARO first performs the K-center clustering algorithm to reduce the number of code elements. The K-center clustering algorithm contains four steps. Firstly, the code elements are sorted according to the sum of fanin and fanout of them. Secondly, the top K code elements are the clustered centers. Thirdly, the nearest code elements of them are clustered into centers. Fourthly, if the number of code lines is less than the threshold, ARO performs the first three steps iteratively. The purpose of performing the K-center algorithm contains the following aspects. Firstly, the number of code elements is reduced. Secondly, code elements are clustered around the K centers, and the K centers are the core code, that is, code elements are clustered around the core code.

After performing the K-center algorithm, the K clusters are used to construct the $k * k$ structure, and $L(K)$ denotes the level of the k_{th} cluster, and the distance between the cluster (r) and the cluster (s) is represented as $d[(r), (s)]$. The details of the clustering process are as follows.

1) There are k clusters in the structure D , and each cluster is composed by a code entity. Let the number m is 0, and the $L(m)$ is 0.

2) The minimum of the distance in the D is represented as min , and the cluster is (r) and (s) .

3) (r) and (s) are clustered together into a new code entity (r, s) . Let the number m is $m + 1$, and the $L(m)$ is $d[(r), (s)]$.

4) The D is updated, and the columns and rows of (r) and (s) are deleted. The new cluster (r, s) is added into D . Let the distance between cluster $k - 1$ and (r, s) is $mind[(k), (r)], d[(k), (s)]$.

5) Step 2 to step 4 are performed iteratively until the number of code lines of the code elements is greater than the threshold.

The K-center-hierarchy clustering algorithm is shown in Algorithm 1.

C. Optimizing architecture

The first step of optimizing architecture is clustering code elements based on the structural similarity. The structural similarity between the code elements is equal to the average of the fanin and the fanout. $RelativeSim(a, b)$ denotes the structural similarity between the code element a and the code element b . The formula is as follows:

$$RelSim(a, b) = \begin{cases} \frac{C \sum_{i=1}^{|O(a)|} \sum_{j=1}^{|O(b)|} RelSim(O_i(a), O_j(b))}{|O(a)||O(b)|} & a \neq b \\ 1 & a = b \end{cases} \quad (4)$$

where $O(a)$ denotes the fanout of a , the parameter C is a damping factor. If $O(a) = O(b) = A$, then $RelSim(a, b) = C * RelSim(A, A) = C$, so $C \in (0, 1)$. If two the structural similarity between two nodes is higher than the threshold, the two nodes are clustered into a new node.

The second step is renaming components. The file name and the directory name are set by developers based on the function of the source code, so we rename components based on their corresponding directory names. The renaming process

Algorithm 1 The K-center-hierarchy clustering algorithm

Input:

Preprocessed file dependency graph $PFDG$

Output:

Component dependency graph CDG

```

1: Procedure Cluster(PFDG graph)
2: for each  $node \in nodesets$  do
3:   if node is a central node then
4:     process the next node
5:   end if
6:   if  $node.scale > Value$  then
7:     process the next node
8:   end if
9:   for  $j=i+1:j \leq nodesets.num:j++$  do
10:    if  $min > s(i, j)$  then
11:       $min = s(i, j)$ 
12:    end if
13:    select the min distance  $s(i, r) = min$ 
14:    combine  $(i)node$  and  $(r)node$  into one cluster
15:    Update graph by updating rules(including delete  $r$  node and update  $nodesets$ )
16:    if node is the latest node then
17:      break
18:    else
19:       $i=i-1$ 
20:      Until all the clusters beyond the cluster size
21:    end if
22:  end for
23: end for

```

contains the following steps: a) Identify the corresponding component of each directory. b) All files are traversed to find out whether there is a specific directory contains most files of the component, if there is, then the component is named the directory name. c) All code elements are traversed to find out whether there are three generations of relatives that contain most of the files in the component, if there is, then the component is named as their common grandparent directory.

IV. EXPERIMENTS AND EVALUATION

In this section, we evaluate ARO by comparing it with some related methods to answer the following research questions:

RQ1: Does our method improve the accuracy of the recovered architecture?

RQ2: Does our method improve the efficiency of the recovered architecture?

A. Accuracy evaluation

MoJoSim is an indicator of evaluating the similarity between the recovered architecture and the ground-truth architecture, and it is used in much related work [11], [12]. In this paper, we use it to evaluate the accuracy of ARO. MoJoSim is calculated as follows.

$$MoJoSim(RA, GA) = (1 - \frac{mno(RA, GA)}{n}) * 100\% \quad (5)$$

where RA is the recovered architecture, GA is the ground-truth architecture, $mno(A, B)$ is the minimum number of *Move* and *Join* operations of converting A to B , n is the number of clustered code elements. If MoJoSim is 100%, it indicates that the two architectures are the same, and 0% indicates that the two architectures are completely inconsistent.

Table I shows the MoJoSim range of the nine methods and our method.

TABLE I
THE MOJOSIM RANGE OF EACH METHOD

Data source	Method	MoJoSim
Our paper	Our method	0.55-0.75
Bittencourt and Guerrero et al. [13]	EQ	0.20-0.60
	KM	0.40-0.80
	MQ	0.30-0.70
	DSM	0.35-0.75
Wu et al. [17]	CL90	0.40-0.48
	CL75	0.45-0.52
	ACDC	0.28-0.35
	SL75	0.10-0.15
	SL90	0.07-0.10

As shown in Table I, our method has a higher MoJoSim value than other methods. DSM, KM, and our method cluster elements based on the distance, but only KM and our method cluster elements around the core code. In a word, the architecture recovered by our method is closer to the ground-truth architecture, so the accuracy of our method is higher than the above methods.

B. Efficiency evaluation

Bittencourt et al. [12] proposed a cluster-based architecture recovery method, then Michele et al. [13] introduced fold-in and fold-out based on Bittencourt's method. Both of the above methods have been widely recognized, so we compare our method with the above two methods to evaluate efficiency of ARO.

PostgreSQL is an open source project, and it is often used as an experimental case for architecture recovery. We choose 30 versions as the experimental cases. The time consumed of the above three methods is shown in Figure 1.

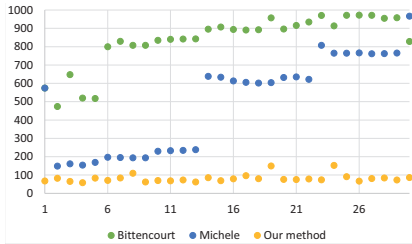


Fig. 1. The time consumed of the three methods

As shown in Figure 1, the average time consumed of the above three methods is 835.288 seconds, 495.568 seconds and 81.814 seconds, respectively. Bittencourt's method takes the longest time, and our method takes the shortest time. In a word, compared to the two methods, our method has higher efficiency.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a method for recovering and optimizing software architecture based on source code and directory hierarchies to improve accuracy and efficiency. ARO

extracts specific information about the implementation of architecture from source code and directory hierarchies to improve accuracy. Then ARO improves the efficiency by preprocessing the file dependency graph and the K-center-hierarchy algorithm. Finally, ARO optimizes architecture. Experimental results indicate that compared with the representative architecture recovery methods listed in this paper, the architectures recovered by ARO has higher accuracy with higher efficiency.

In our future work, we will extract more information about architecture to improve accuracy, such as the compiled build files, architecture documents, and other files.

ACKNOWLEDGEMENTS

This work is supported in part by the National Key R&D Program of China under Grant 2018YFB1003902, in part by the Cooperation Project with Huawei Technologies Co., Ltd., under Grant YBN2016020009, and in part by National Natural Science Foundation of China under Grant 61872078, Grant 61572126, and Grant 61402103.

REFERENCES

- [1] Fabian Brosig, Philipp Meier, Steffen Becker, Anne Kozirolek, Heiko Kozirolek, and Samuel Kounev. Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures. *IEEE Transactions on Software Engineering*, 41(2):157–175, 2015.
- [2] Ricardo Britto, Daria Smitte, and Lars Ola Damm. Software architects in large-scale distributed projects: An ericsson case. *IEEE Software*, 33(6):48–55, 2016.
- [3] Kozirolek, Heiko, Schlich, Bastian, Becker, Steffen, Hauck, and Michael. Performance and reliability prediction for evolving service-oriented software systems. *Empirical Software Engineering*, 18(4):746–790, 2013.
- [4] Michele Risi, Giuseppe Scanniello, and Genoveffa Tortora. *Using fold-in and fold-out in the architecture recovery of software systems*, volume 24. 2012.
- [5] Xianglong Kong, Bixin Li, Lulu Wang, and Wensheng Wu. Directory-based dependency processing for software architecture recovery. *IEEE Access*, 6:52321–52335.
- [6] Liu Jing, Zhiming Lui, Xiaoshan Li, Jifend He, and Yifeng Chen. Towards the integration of a formal object-oriented method and relational unified process. *Software Evolution with Uml & Xml*, pages 101–133, 2005.
- [7] Fritz Solms. Experiences with using the systematic method for architecture recovery (symar). In *South African Institute for Computer Scientists and Information Technologists Conference*, pages 170–178, 2013.
- [8] Onaiza Maqbool and Haroon Babri. Hierarchical clustering for software architecture recovery. *IEEE Transactions on Software Engineering*, 33(11):759–780, 2007.
- [9] Kamran Sartipi. Software architecture recovery based on pattern matching. In *International Conference on Software Maintenance*, pages 293–296, 2003.
- [10] H Sajjani. Automatic software architecture recovery: A machine learning approach. In *IEEE International Conference on Program Comprehension*, pages 265–268, 2012.
- [11] Thibaud Lutellier, Devin Chollack, Joshua Garcia, Lin Tan, Derek Rayside, Nenad Medvidovic, and Robert Kroeger. Comparing software architecture recovery techniques using accurate dependencies. In *IEEE/ACM IEEE International Conference on Software Engineering*, pages 67–69, 2015.
- [12] Roberto Almeida Bittencourt and Dalton Dario Serey Guerrero. Comparison of graph clustering algorithms for recovering software architecture module views. In *European Conference on Software Maintenance & Reengineering*, pages 251–254, 2009.
- [13] Anna Corazza, Sergio Di Martino, and Giuseppe Scanniello. A probabilistic based approach towards software system clustering. In *European Conference on Software Maintenance and Reengineering*, pages 88–96, 2011.

Automated user-oriented description of emerging composite ambient applications

M. Koussaifi, S. Trouilhet, J.-P. Arcangeli, J.-M. Bruel
IRIT, University of Toulouse, France

Abstract—Ambient environments consist of components surrounding the user and offering services. Applications can here be composed opportunistically and automatically by an intelligent system that puts together available components. Thus, applications that are a priori unknown emerge from the environment. The problem is in the intelligible presentation to an average user of those emerging composite applications. Our approach consists in automatic generation of user-oriented application descriptions from unit descriptions of each component and service. For that, we propose a well-defined language for component description and a method for combining descriptions. A prototype has been developed and used to experiment the generation of different composite application descriptions. Based on these experiments, we assess the degree of fulfillment of the requirements we have identified for the problem.

I. INTRODUCTION

Applications of the Internet of Things, ambient and cyber-physical systems consist of fixed or mobile connected devices. Devices host independently developed and managed software components that provide services specified by interfaces and, in turn, may require other services [1]. Components are building blocks that can be assembled by binding required and provided services to build composite applications.

Due to mobility and separate management, devices and software components may appear and disappear without this dynamics being foreseen. Hence, the environment is open and its changes are out of control. Humans are at the core of these dynamic systems and can use the applications at their disposal. Ambient intelligence aims at offering them a personalized environment adapted to the current situation, anticipating their needs and providing them the right applications at the right time with the least effort possible.

We are currently exploring and designing a solution in which components are dynamically and automatically assembled to build new composite applications and so customize the environment at runtime. Our approach is rather disruptive: unlike the traditional goal-directed top-down mode, applications are built on the fly in bottom-up mode from the components that are present and available at the time, without user needs being made explicit. That way, composite applications continuously emerge from the environment, taking advantage of opportunities as they arise: for example, a slider on a smartphone can opportunistically be composed with a connected lamp and provide

the user with a lightening service when entering a room. Here, contrary to the traditional SOA paradigm, the user does not specify a service or search for it in “pull mode”, but context-adapted applications are provided in “push mode”.

Automated composition is supported by an assembly engine in line with the autonomic computing principles and the MAPE-K model [2]: it senses the existing components and decides of the connections (it may bind a required service and a provided one if their interfaces are compatible) without using a pre-established plan. The heart of the engine is a distributed multi-agent system where agents, close to the software components and their services, cooperate and decide on the connections [3]. To make the right decisions and offer relevant applications, the engine (*i.e.*, the agents) learns at runtime by reinforcement. Thus, the engine assures proactivity and runtime adaptation in the context of openness, dynamics and unpredictability.

In such a context of automation based on artificial intelligence, we believe that, whatever the engine's decisions are, the deployment of emerging applications should remain under user control. So, she/he must first be informed of the new application. Then, depending on its interest, she/he must be able to accept it or not, possibly to modify it (provided that she/he has the required skills) and so to contribute to the customization of her/his environment. So, the user must be put “in the loop” [4]. In addition, the user's actions about the emerging application (acceptance, rejection, modification) are sources of feedback for the engine's learning. Based on them, the engine builds a model of the user's preferences and habits. Unknown a priori, this model is built at runtime and evolves dynamically.

Therefore, it is essential to assist the user in the appropriation of the emerging applications pushed by the engine. For that, applications must be presented to the user in a useful and understandable way. The goal of this paper is to propose a solution to provide the user with an intelligible description of emerging applications.

The paper is organized as follows. Section II describes in more details the problem and lists the main requirements. Section III analyzes the related work on service description and shows that the solutions are very limited in relation to our problem. Section IV presents and illustrates our approach to meet the requirements. Section V describes an experiment based on a prototype we have developed. It shows the feasibility of our approach and assesses whether it meets the requirements. Finally, a conclusion is given in Section VI as well as the perspectives of this work.

II. PROBLEM STATEMENT

In the absence of prior specification, emerging applications are unknown a priori and possibly surprising. They result from local interactions between distributed agents that constitute the engine. Composition relies on learned user preferences and a matching between required and provided services.

The user must be aware of the emerging application, its function and how to use it, to consider if she/he could benefit from. Therefore, applications must be presented in an intelligible way. Here, we target average users that are not familiar to programming and computer science. For instance, the user may be the inhabitant of a smart house or a public transport traveller in a smart city.

Consider a simple assembly consisting of a switch and a lamp. In that case, we would ideally like to tell the user something like “if you click on the switch, the lamp will turn ON/OFF”. Therefore, the problem lies in the construction of the understandable description of an application defined by an assembly of software components, and to compute the description from the participating components, their services and bindings.

A. Previous work

In [4], we have proposed an architecture that puts the user in the loop (see Fig. 1). An editor presents the emerging application, allows the user to accept it, then it is deployed, or to reject it, then it is cancelled, or to modify it (that is add, remove or change bindings between services). Acceptance, rejection, and modifications are notified back to the engine for learning.

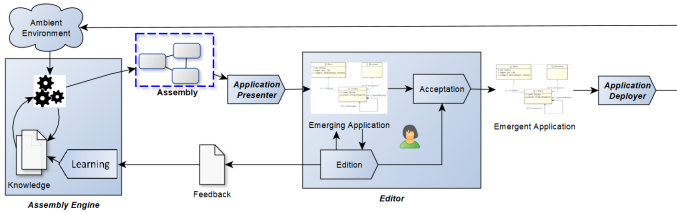


Fig. 1. Overall solution architecture

For that, the editor exposes a structural description of the application (see Fig. 2) that is an editable graph of software components that are connected through their respective services, as well as other available components that may be useful. This is achieved using model transformation techniques that transform the output of the engine (set of components, services and bindings) into a model (conforming to a metamodel we have defined for this purpose) that is presented for the user.

In the state of our work, the solution is limited to the presentation of an editable structure of the application. On one hand, this allows the user to build of a tailor-made ambient environment. On the other hand, this requires to understand component-based architecture, at least the meaning of an assembly of components. Therefore, structural presentation is not understandable by an average user,

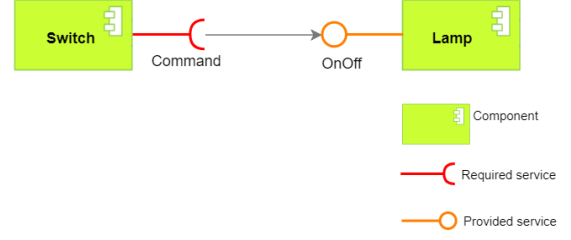


Fig. 2. Structural presentation via the editor

who needs to know the function rendered by the application (in other words its semantics), and how to use it.

The next section lists the requirements we have identified for an efficient presentation of an emerging application.

B. Requirements

1) **Semantics:** The function (*i.e.*, the semantics) of the application must be exhibited. For example, “the application allows to light up the lamp”.

2) **Usage:** The instructions on how to use the application must be exhibited. For example, “press the switch to turn ON/OFF the light”.

3) **Intelligibility:** The description must be understandable by an average user, without programming skills.

4) **Presentation scalability:** The description should remain useful and intelligible even when the application has about ten or more components.

5) **Automated processing:** The description must be automatically built by combining unit descriptions of components without human support.

6) **Expressiveness:** The description language must be expressive enough for software engineers with standard skills to make descriptions of the components they provide.

III. RELATED WORK

Fundamentally, software components and services are developed and documented for composition and reuse. In practice, they are built from scratch or from existing ones. In service-oriented engineering, service discovery and selection are fundamental operations. Selection generally aims to choose a service on a qualitative basis among several ones that have been previously discovered. Discovery and selection are performed either manually or more or less automatically, at design or runtime, from service descriptions.

In this section, we examine the related work in the field of service description. We first study the questions related to the target and objective of the description, then the questions of content and tools of description.

A. For whom and why describing a service?

Designers use service descriptions as documentation. They likewise describe the intent and use of the services they develop. When engineers specify business processes to be realized through the composition of existing services,

they describe (composite) services too. For example, composition of Web services are first explicitly specified by the service requester, then processed more or less automatically [5]. Thus, in that top-down mode, the demanded composite service is specified a priori, so no more description is necessary.

In [6], authors propose a user-centric service composition platform that assists end-users without skills in service-oriented engineering. End-users first enter their goals using a few keywords. Then an editor presents the available services and suggests possible and modifiable processes.

Most of the existing solutions use service description to support automated service discovery, selection and composition. Services are described to be processed by a program. Description allows service location and use, as is the case for WSDL [7] in the field of Web Services.

Semantic description of Web services first targets interoperability [8]. Relying on semantics also has a positive impact on the quality of the composition [9], in particular when Quality of Service (QoS) attributes are considered [10], [11]. In [12], authors propose semantic enhancement of software components with their properties and functionality to support matching between candidate components.

B. How to describe a service?

Service description can take more or less advanced forms depending on the requirements for discovery, selection and composition. In [13], authors overview and classify service description approaches used in automated service composition research.

In a basic way, descriptions may be limited to a syntactic level. For example, in object-oriented middleware like Java RMI [14], remote objects that provide services are registered and located only through a name. Services (resources in general terms) may be described more precisely using keywords in order to be retrieved by their characteristics rather than a simple identifier.

The semantic description of a service can be functional. It can take the form of a signature with inputs and outputs, possibly completed by preconditions and effects [15]. Authors of [11] explain that signature is not enough because different functions may have the same signature on the one hand, and that two services rendering the same function may differ fundamentally in their performances on the other hand. Therefore, service description may include extrafunctional properties that is QoS-related properties.

In [16], authors have proposed different techniques to create descriptions of services using the DAML-S language that was proposed to bridge the gap between the Web services infrastructure and the Semantic Web [17]. According to [13], OWL-S that succeeded to DAML-S has become a standard for industrial service composition. OWL-S [18] is an ontology for describing Semantic Web services that enables their automated discovery, composition and use. Ontology-driven description of services proved to be efficient for selection and composition [6].

C. Analysis

There are many solutions for functional and extrafunctional service description. Most of them focus on service discovery, selection, and top-down composition in order to build a complex service from unit ones. In our bottom-up approach, as the complex service to be built is unknown, there exists no solution which aims at combining descriptions. In most cases, service description supports automation, for example when based on ontologies. But in that case, descriptions are little or not at all intelligible by average human users. In addition, when extrafunctional properties are considered, they mainly concern the quality of services, but not their usage. In conclusion, to the best of our knowledge, there is no work that meets our requirements, mainly those concerning usage, intelligibility, and automated processing, in the context of bottom-up and goal-free application construction. Nevertheless, a functional description of services using signatures with preconditions and effects [15] may help in extracting the semantic information about the components' behavior and their interactions, that should be useful for the entire application description.

IV. PROPOSITION

This section presents our approach to describe components and their services, and to compute user-oriented descriptions of emerging composite ambient applications (assemblies of software components). Descriptions mainly consist of rules that explain the components and the applications. Composite application descriptions are generated from the unit descriptions of the components that are given at component design time, and the bindings between services that are supplied by the engine. Generation is achieved by combining the descriptions together, precisely the rules that belong to each unit description. At the end, the combination process aims at building a rule or a set of rules that describes the application, that can be then transformed into a text readable by the user. Our contributions are: (i) a language for the description of components' services and (ii) a combination method.

In the following, our proposition is explained in details.

A. Component and Service Description

A component description (CD) is a tuple that expresses how the component and its services work and interact with other connected components.

$$CD = \langle \text{ComponentName}, \text{Role}, \text{States}, \text{ProvidedServices}, \text{RequiredServices} \rangle$$

ComponentName and *Role* are strings: the name of the component and a free text (e.g., *ComponentName* = "Switch", *Role* = "Send a signal when pressed"). As components may have an internal state, such as a lamp that is ON/OFF, *States* is the (possibly empty) set of possible states (e.g., *States* = {"ON", "OFF"}). Last, the component's

required and provided service descriptions are gathered in the *ProvidedServices* and *RequiredServices* sets.

A service, whether provided or required, is also described by a tuple (SD).

$$SD = \langle \text{ServiceName}, \text{IOAction}, \text{Launcher}, \text{ServiceDescription}, \text{BoundTo}, \text{Rules} \rangle$$

ServiceName is a string (e.g., *ServiceName* = "Command").

IOAction represents how the service interacts with other services. It may have the following forms: *VAL@OUTPUT* or *TRIGGER ServiceName*. The first form refers to the emission of a message on the output interface of a required service. *VAL* covers all possible data types that the service handles (as the services have previously been composed by the engine, the type matching problem has been already resolved; thus types are useless for our descriptions), and may even be omitted. The second form refers to the transfer of control between services inside a component. Furthermore, *IOAction* can be empty for a provided service handling only the evolution of the component's state, without any output (e.g., the *OnOff* provided service of the lamp that changes the state to ON/OFF).

Launcher is a key defining what activates the service. It covers two cases. The first refers to an external interaction coming from another component (*onRequired*) or to an internal one coming from the component itself (*onTriggered*). The second case refers to an interaction coming from the user and can have multiple values such as *onButtonPressed*, *onSliderDragged*, *onCheckBoxChecked*...

ServiceDescription describes the service in a free text (e.g., *ServiceDescription* = "Turn ON/OFF the lamp"). This attribute is used by our combination algorithm to generate a textual form of the description (see section IV-B).

BoundTo is a set of services within their component (C-S) described as follows.

$$C - S = \text{ComponentName.ServiceName}$$

For example, $C - S = \text{Switch.Command}$.

Rules is a set of logic rules of the form "*Condition* \Rightarrow *Consequence*" that describes the service behavior. It is written as follows in BNF notation [19] where $\langle cp \rangle$ is a comparator, $S \in \text{States}$ and V is a value. Note that *IOAction* and *Launcher* have been put out of the rules for expressiveness and separation of concerns purpose.

$$\begin{aligned} & \text{Launcher} \\ & [\wedge (\text{STATE} < cp > S)] \\ & [\wedge (\text{VAL@INPUT} < cp > V)] \\ & \Rightarrow \\ & \text{IOAction} \mid \\ & \text{STATE} = S \mid \\ & \text{IOAction} \wedge (\text{STATE} = S) \mid \\ & \text{NOP} \end{aligned}$$

The common case is "*Launcher* \Rightarrow *IOAction*". Premises concerning the component's state and the inputted value are optional. It is usually the case for services that have no condition to check, other than their *Launcher*, before triggering their action. For example, for a switch to issue a command, it is only necessary that the button is pressed by the user.

In the general case, the condition part of a rule may contain a test on the component's state or on an inputted value. For the consequence part of the rule, several forms are possible. In particular, it may contain a state changing operation. Furthermore, *NOP* is a special key used if the service does not carry out any operation.

Here are examples of service descriptions of the *Command* service required by a switch and the *OnOff* service provided by a lamp, that have been connected by the assembly engine. Component and service descriptions have initially been written by the designer. They are completed by the engine by filling the *BoundTo* attribute according to the emerging assembly.

$$\begin{aligned} & \langle (\text{ServiceName}) \text{Command}, \\ & \quad (\text{IOAction}) @\text{OUTPUT}, \\ & \quad (\text{Launcher}) \text{onButtonPressed}, \\ & \quad (\text{ServiceDescription}) \text{Send a signal}, \\ & \quad (\text{BoundTo}) \{\text{Lamp.OnOff}\}, \\ & \quad (\text{Rules}) \{\text{Launcher} \Rightarrow \text{IOAction}\} \rangle \end{aligned}$$

$$\begin{aligned} & \langle (\text{ServiceName}) \text{OnOff}, \\ & \quad (\text{IOAction}), \\ & \quad (\text{Launcher}) \text{onRequired}, \\ & \quad (\text{ServiceDescription}) \text{Turn ON/OFF the lamp}, \\ & \quad (\text{BoundTo}) \{\text{Switch.Command}\}, \\ & \quad (\text{Rules}) \{\text{Launcher} \wedge (\text{STATE} == \text{OFF}) \\ & \quad \quad \Rightarrow \text{STATE} = \text{ON}, \\ & \quad \quad \text{Launcher} \wedge (\text{STATE} == \text{ON}) \\ & \quad \quad \Rightarrow \text{STATE} = \text{OFF}\} \rangle \end{aligned}$$

The next section presents the method for combining descriptions.

B. Combination of descriptions

Application descriptions are generated mainly from the rules that describe the services, and then from the remaining attributes (if used by the rules). If a service S1 is connected to a service S2 then the rules of S1 are combined with the rules of S2 to generate the rules that describe the composition.

The combination algorithm first finds matching keys available in each possible pair of rules which belong to S1 and S2 descriptions. For example: *VAL@OUTPUT* matches *onRequired* and *VAL@INPUT*; *TRIGGER Service-Name* matches *onTriggered*. Then, the algorithm infers the combined rules by transitivity. For example:

```
* R1: A  $\Rightarrow$  B  $\wedge$  C
* R2: C  $\Rightarrow$  D
R1 and R2 are combined into:
* R: A  $\Rightarrow$  B  $\wedge$  D
```

Note that there might be several combined rules at the same time, for example:

```
* R1: A  $\Rightarrow$  B
* R2: B  $\wedge$  B'  $\Rightarrow$  C
* R3: B  $\wedge$  B''  $\Rightarrow$  D
R1, R2 and R3 are combined into:
* R': A  $\wedge$  B'  $\Rightarrow$  C
* R'': A  $\wedge$  B''  $\Rightarrow$  D
```

Here is an example for the Switch-Lamp application:

```
* R': LauncherCommand  $\wedge$  (STATE == ON)  $\Rightarrow$  STATE = OFF
* R'': LauncherCommand  $\wedge$  (STATE == OFF)  $\Rightarrow$  STATE = ON
```

The combined rules are transformed into a textual form to be presented to the user. An attribute is linked to its component and the label is replaced by its content (*Launcher_{Command}* becomes *OnButtonPressed of Switch*). The *ServiceDescription* content is used to make the elements of the rule more explicit (the *STATE* element and *Turn ON/OFF the lamp* are compared). Finally, the \Rightarrow is translated in a verbal form (*IMPLIES ... IF*). In addition, the algorithm is able to group rules that have the same launcher. The textual presentation of the Switch-Lamp application is:

```
onButtonPressed of Switch IMPLIES
  Turn OFF the lamp IF Lamp is ON
  Turn ON the lamp IF Lamp is OFF
```

Note that the generated textual description may sometimes not be grammatically correct. At this point, syntax improvement is left for future work.

The above example has the simplest topology, but the solution targets more complex ones: pipeline, star... Our description language can easily be extended. For example in order to cover the case of a component that requires several services sequentially, a sequence operator could be added to the description language and handled by the combination algorithm.

The next section shows the experimentation we have carried out and analyzes our solution in relation to the requirements.

V. EXPERIMENTATION AND ANALYSIS

A. Proof of concept

In order to demonstrate the feasibility of our approach, we have developed a prototype in Java, where each component and its services XML-like descriptions are stored in a separate file. It was tested on different composite applications built by our assembly engine.

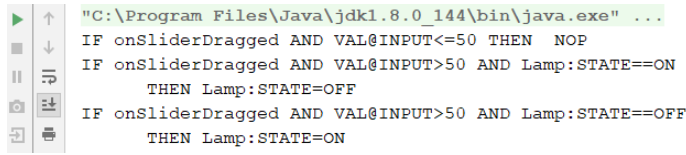
Here is an example with three components assembled in pipeline mode: a slider, a converter and a lamp. The slider acts as a switch. It requires the *ProcessVal* service. The converter provides the *Transform* service: it receives a value and, if greater than 50, transforms it into an order for the lamp through the *Order* required service. As in the previous section, the lamp provides the *OnOff* service. The descriptions of the services are given below (see Section IV-A for *OnOff*).

```
< (ServiceName) ProcessVal,
  (IOAction) VAL@OUTPUT,
  (Launcher) onSliderDragged,
  (ServiceDescription) Send a value  $\in$  [0,100],
  (BoundTo) {Converter.Transform},
  (Rules) {Launcher  $\Rightarrow$  IOAction} >
```

```
< (ServiceName) Transform,
  (IOAction) TRIGGER Order,
  (Launcher) onRequired,
  (ServiceDescription) Change value into signal,
  (BoundTo) {Slider.ProcessVal}
  (Rules) {Launcher  $\wedge$  (VAL@INPUT > 50)
     $\Rightarrow$  IOAction,
    Launcher  $\wedge$  (VAL@INPUT  $\leq$  50)  $\Rightarrow$  NOP} >
```

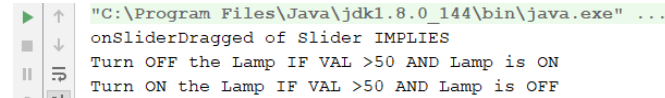
```
< (ServiceName) Order,
  (IOAction) @OUTPUT,
  (Launcher) onTriggered,
  (ServiceDescription) Send a signal,
  (BoundTo) {Lamp.OnOff},
  (Rules) {Launcher  $\Rightarrow$  IOAction} >
```

Fig. 3 shows the rules resulting from the combination algorithm. Then, the rules are transformed into a more intelligible textual version to describe the emerging application (Fig. 4).



```
"C:\Program Files\Java\jdk1.8.0_144\bin\java.exe" ...
IF onSliderDragged AND VAL@INPUT<=50 THEN NOP
IF onSliderDragged AND VAL@INPUT>50 AND Lamp:STATE==ON
    THEN Lamp:STATE=OFF
IF onSliderDragged AND VAL@INPUT>50 AND Lamp:STATE==OFF
    THEN Lamp:STATE=ON
```

Fig. 3. Description's rules of the emerging application



```
"C:\Program Files\Java\jdk1.8.0_144\bin\java.exe" ...
onSliderDragged of Slider IMPLIES
Turn OFF the Lamp IF VAL >50 AND Lamp is ON
Turn ON the Lamp IF VAL >50 AND Lamp is OFF
```

Fig. 4. User-oriented description of the emerging application

B. Analysis

Rules describing the composite application are actually inferred. They provide the information about both the function of the application and how the user can interact with it. Thus, the rule-based description of the components and their services in an assembly makes possible to satisfy the main requirements we have defined (see Section II-B) concerning *semantics*, *usage*, and *automated processing*. By transforming rules into text, the understanding is made easier. Nevertheless, no real users have yet assessed the *intelligibility*. User assessment experiments could help us in improving the description language and the rule combination process. Concerning *expressiveness*, we consider that the rule-based language is expressive enough for software engineers. Moreover many elements of CD and SD could be automatically extracted from the code of components, in particular from the signatures of methods. *Presentation scalability* requirement has yet to be improved. We would like to allow the folding and unfolding of descriptions when a number of components are involved, and therefore offer a kind of "responsive" presentation of applications with different levels of abstraction.

VI. CONCLUSION

In this paper, we have exposed an approach that aims to answer the requirements to generate user-oriented intelligible descriptions of emerging assemblies of software components. We have presented the limitations of the current solutions and highlighted the benefits of our one. We have developed a proof of concept that shows that our approach can meet the requirements. Further experiments must now be carried out on more complex composite applications and topologies in order to consolidate our solution and enrich the description language. Real users should be involved in the experiments to improve and validate intelligibility and scalability of the presentation.

Our next step towards addressing the scalability issue will be to fully use the power of Model-Driven Engineering

(MDE) approaches and tools to support the automatic generation of combination algorithms from the description language definition itself. Our description language being a domain-specific language (DSL), and our input assembly being a model, MDE which has been proved useful in this particular case [20] will allow us to define transformation between assemblies and their descriptions.

REFERENCES

- [1] I. Sommerville, "Component-based software engineering," in *Software Engineering*, ch. 16, pp. 464–489, Pearson Education, 10 ed., 2016.
- [2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41–50, Jan. 2003.
- [3] W. Younes, S. Trouilhet, F. Adreit, and J.-P. Arcangeli, "Towards an intelligent user-oriented middleware for opportunistic composition of services in ambient spaces," in *Proceedings of the 5th Workshop on Middleware and Applications for the Internet of Things, M4IoT'18*, (New York, NY, USA), pp. 25–30, ACM, 2018.
- [4] M. Koussaifi, S. Trouilhet, J.-P. Arcangeli, and J.-M. Bruel, "Ambient intelligence users in the loop: Towards a model-driven approach," in *Software Technologies: Applications and Foundations* (M. Mazzara, I. Ober, and G. Salaün, eds.), pp. 558–572, Springer, 2018.
- [5] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decade's overview," *Information Sciences*, vol. 280, pp. 218 – 238, 2014.
- [6] H. Xiao, Y. Zou, R. Tang, J. Ng, and L. Nigul, "Ontology-driven service composition for end-users," *Service Oriented Computing and Applications*, vol. 5, p. 159, Mar 2011.
- [7] "Web Services Description Language." <https://www.w3.org/TR/wsdl/>. Accessed: 2019-01-31.
- [8] H. Nacer and D. Aissani, "Semantic web services: Standards, applications, challenges and solutions," *Journal of Network and Computer Applications*, vol. 44, pp. 134–151, Sept. 2014.
- [9] Y. Charif and N. Sabouret, "An overview of semantic web services composition approaches," *Electronic Notes in Theoretical Computer Science*, vol. 146, no. 1, pp. 33 – 41, 2006. Proceedings of the First International Workshop on Context for Web Services (CWS 2005).
- [10] E. Chindenga, M. S. Scott, and C. Gurajena, "Semantics Based Service Orchestration in IoT," in *Proceedings of the South African Institute of Computer Scientists and Information Technologists, SAICSIT'17*, (New York, NY, USA), pp. 7:1–7:7, ACM, 2017.
- [11] A. Hurault, F. Camillo, M. Daydé, R. Guivarch, M. Pantel, C. Puglisi, and H. Astsatryan, "Semantic description of services: issues and examples." Computer Science and Information Technologies, Yerevan (Arménia), 2009.
- [12] J. M. Gomez, S. Han, I. Toma, B. Sapkota, and A. Garcia-Crespo, "A Semantically-enhanced Component-based Architecture for Software Composition," in *Int. Multi-Conf. on Computing in the Global Information Technology (ICCGI'06)*, pp. 43–47, Aug 2006.
- [13] Y. Fanjiang, Y. Syu, S. Ma, and J. Kuo, "An overview and classification of service description approaches in automated service composition research," *IEEE Transaction on Services Computing*, vol. 10, pp. 176–189, March 2017.
- [14] "Java Remote Method Invocation (RMI)." <https://docs.oracle.com/javase/tutorial/rmi/index.html>. Accessed: 2019-01-31.
- [15] M. Klusch, "Semantic web service description," in *CASCOM: Intelligent Service Coordination in the Semantic Web*, (Basel), pp. 31–57, Birkhäuser Basel, 2008.
- [16] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing web services on the semantic web," *The VLDB Journal*, vol. 12, pp. 333–351, Nov. 2003.
- [17] M. Paolucci and K. Sycara, "Autonomous semantic web services," *IEEE Internet Computing*, vol. 7, pp. 34–41, Sept. 2003.
- [18] "OWL-S: Semantic Markup for Web Services." <https://www.w3.org/Submission/OWL-S/>. Accessed: 2019-01-31.
- [19] "Backus-Naur Form (BNF)." <https://www.w3.org/Notation.html>. Accessed: 2019-01-31.
- [20] H. Bruneliere, R. Eramo, A. Gomez, V. Besnard, J.-M. Bruel, M. Gogolla, A. Kästner, and A. Rutle, "Model-Driven Engineering for Design-Runtime Interaction in Complex Systems: Scientific Challenges and Roadmap," in *MDE@DeRun 2018 workshop*, vol. 11176 of LNCS, June 2018.

Usability of Chatbots: A Systematic Mapping Study

Ranci Ren

Dep. Ing. Informática
Univ. Autónoma de Madrid
Madrid, Spain
ranci.ren@estudiante.uam.es

John W. Castro

Dep. Ing. Informática y Ciencias de la Computación
Universidad de Atacama
Copiapó, Chile
john.castro@uda.cl

Silvia T. Acuña, Juan de Lara

Dep. Ing. Informática
Univ. Autónoma de Madrid
Madrid, Spain
{silvia.acunna, juan.delara}@uam.es

Abstract— Background: The use of chatbots has increased considerably in recent years. These are used in different areas and by a wide variety of users. Due to this fact, it is essential to incorporate usability in their development. Aim: Our objective is to identify the state of the art in chatbot usability and applied human-computer interaction techniques, to analyze how to evaluate chatbots usability. Method: A systematic mapping study has been conducted, searching the main scientific databases. The search retrieved 170 citations and 19 articles were retained as primary studies. Results: The works were categorized according to four criteria: usability techniques, usability characteristics, research methods and type of chatbots. Conclusions: Chatbot usability is a very incipient field of research, where the published studies are mainly surveys, usability tests, and rather informal experimental studies. Hence, it becomes necessary to perform more formal experiments to measure user experience, and exploit these results to provide usability-aware design guidelines.

Keywords—Usability; chatbots; systematic mapping study

I. INTRODUCTION

Chatbots are computer programs with a textual or voice interface, based on natural language [1]. They are specifically designed to make user interaction as natural as possible, and have received extensive attention from academia and industry in recent years. Chatbots not only enable a faster and more natural way to access information, but they will become a key factor in the process of humanizing machines in the near future.

Usability is defined as the degree to which a program can be used to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a specified context of use [2]. Usability is a critical aspect in interactive software systems and so it is essential to incorporate usability in chatbots, to improve user experience. Chatbots are become pervasive and are used in many areas, such as bookings of all sorts of services, to obtain medical advice and for online shopping [1][3][4]. The multiple uses and benefits of chatbots explain their strong growth in terms of users, satisfaction and saving resources. It is expected that the number of users will grow in the US by 23.1% [5]. Although the market is still beginning to take shape (compared to the number of websites, the number of bots is still not large) it is estimated that the market size will expand massively [4].

Many universities and commercial companies have put into use chatbots interacting with mature systems. At the commercial level, Facebook messenger already has more than 300,000 chatbots in use [5]. This makes downloading and installing new apps unnecessary, and the use of smartphones allows for personalization possibilities [6]. Further, the use of

chatbots can be more cost-effective than human-assisted support [7]. Some companies are building chatbots independently (e.g., Microsoft is promoting the idea of “*conversation as a Platform*”) to support a variety of media, from Skype to search [8]. Chatbots are not an emerging concept. Research on dialogue systems can be traced back to the 50s, where Alan M. Turing posed the question “*can machines think?*” proposing the Turing test as a criterion for judging whether the machine has intelligence [9]. Weizenbaum’s development of ELIZA at MIT in the 60s can be considered the first dialogue system [1]. Lately, the advances in natural language processing (NLP) have boosted the raise of many chatbot development frameworks (e.g. DialogFlow (<http://dialogflow.com>)).

However, there are currently few works that discuss the usability of chatbots in an integrated and formalized manner. The objectives of our research are to identify the state of the art in chatbots usability and the applied Human-Computer Interaction (HCI) techniques by a Systematic Mapping Study (SMS) and to analyze how to evaluate the usability of chatbots. The contribution of this research is a picture of the current state of usability in chatbots. For this purpose, we present a SMS where we classify the types of chatbots, the measured usability characteristics, the applied usability techniques and the research methods used to evaluate chatbot usability.

Paper organization. In Sec. 2, we present related work. In Sec. 3, we describe the research method of the SMS. Sec. 4 presents the results of the SMS. In Sec. 5, we discuss the results and threats to validity, and finally Sec. 6 concludes.

II. RELATED WORKS

We found only three systematic reviews related to chatbots [9][10][11]. The one by Klopfenstein et al. [9] surveys conversational interfaces, patterns, and paradigms. However, they do not detail the literature retrieval process, and hence it may be potentially incomplete. The survey traces the history of chatbots, from ELIZA to modern chatbots for MOOCs. They conclude that only a subset of chatbots are designed for communicating in natural language, which sometimes makes users disappointed. Then they compare features of major messaging platforms that support bots, like Messenger, WeChat, Line and Skype. Most of them already support a variety of message types, pictures, videos, and sounds. However, none of them have comprehensive enough features. For example Line has groups, buttons and carousel features, but no payment and quick message reply. They detail advantages of bots for users and developers, and conclude

stressing the benefits of chatbots as a new software platform to provide services and data to users.

The work by Ramesh et al. [10] surveys design techniques for conversational agents. The paper presents various solutions for building chatbots, including AIML, NLP and Natural Language Understanding (NLU). The authors describe the general structure of a chatbot, which consists of a responder, classifier and graph master. Then, they list several design techniques for chatbots, from pattern matching, to recurrent neural networks. They stress that NLP techniques are increasingly being used in recent years. The paper presents a classification of chatbots, which includes retrieval-based and generative-based, long and short conversations and open/closed domain. Retrieval-based chatbots pick responses from a pool of predefined ones. The third work, by Laranjo et al. [11] makes a systematic review of conversational agents in health. This review retrieved 1,513 research papers, and identified 17 primary studies. The search was performed in April 2017 and updated in February 2018. They describe 14 different conversational agents distinguishing type of communication technology, dialogue management, dialogue initiative, input modality and task-oriented aspects. The evaluation measures were divided into three main types: technical performance, user experience and health-related measures.

Overall, these works do not focus on usability techniques or usability characteristics of chatbots. Therefore, to the best of our knowledge, there is no SMS on the status of the chatbot usability. Our work covers this gap.

III. RESEACH METHOD

We aim to answer the following research questions: (RQ1) *What is the state of the art of usability in the development of chatbots?* and (RQ2) *How to evaluate the usability of chatbots using HCI principles?* To answer both questions, we have executed an SMS to identify and classify these issues in the published literature [12].

Search String Selection

The first step is identifying search strings and relevant keywords. For this purpose, several options were tried and the best one chosen. In particular, we first read some initial articles, obtaining keywords and basic knowledge related to the topic. After combining the opinions of two experts in HCI, we opted for the search string: (usability OR “usability technique” OR “usability practice” OR “user interaction” OR “user experience”) AND (chatbots OR “chatbots development” OR “conversational agents” OR chatterbot OR “artificial conversational entity” OR “mobile chatbots”).

Databases and Search Protocol

The search was performed in sequence from Scopus, ACM Digital Library, IEEE Xplorer, SpringerLink and Science Direct. The search fields used were determined by the options provided by each database. Considering that the concept of chatbots is still relatively new, the search range is from January 2014 to October 2018. We ordered the search considering the data base that returned most results. The search fields were selected to assure that searches were similar across data bases.

The criteria used to retrieve the fundamental studies are summarized below.

- *Inclusion criteria:* The paper is written in English; AND The abstract or title mentions an issue regarding the chatbots and usability; OR The abstract mentions an issue related to usability engineering or HCI techniques; OR The abstract mentions an issue related to the user experience.
- *Exclusion criteria:* The paper does not present any issue related to the chatbots and usability; OR The paper does not present any issue related to the chatbots and user interaction; OR The paper does not present any issue related to the chatbots and user experience.

Paper Selection

The searches were run using the search string defined. The number of papers returned by the first search was 170, which are called *Retrieved Papers*. Then by inspecting the title, keywords and abstract of each retrieved paper, 41 papers were filtered to the group of *Candidate Papers*. The whole group of Candidate Papers was screened for duplicates. When duplicates were found, only the first occurrence of the paper was counted and maintained, the others were deleted. The final group has 39 papers, which is called *Non-Duplicate Candidate Papers*. Each paper of the Non-Duplicate Candidate Papers group was read, to determine if they described any sort of usability of chatbots. The results were cross-checked by two experts in the HCI area, and any disagreement was discussed and resolved in our meetings. Finally, 19 papers were identified as primary studies.

IV. SYNTHESIS OF THE RESULTS

Figure 1 provides an overview of the primary studies retrieved by the SMS. It is made of three categories, determined by the year of publication, type of paper (conference, journal, chapter) and usability characteristics. The left-hand side is composed of two scatter (XY) charts with bubbles at the intersections of each category. The size of each bubble is determined by the number of primary studies that have been classified as belonging to the respective categories at the bubble coordinates.

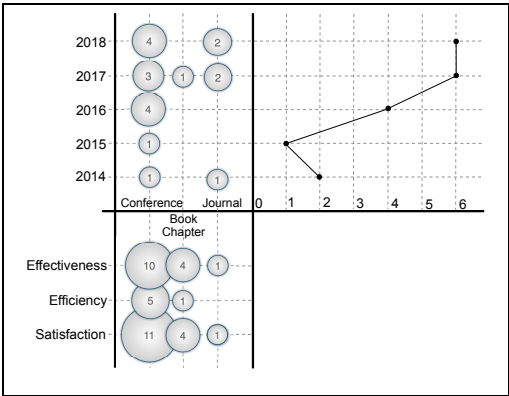


Figure 1. Overview of the Primary Studies

The right-hand side of the Figure shows the number of primary studies by publication year. Publications started to grow from 2015, and many articles (mainly in conferences)

have been published each year since then, confirming the interest in the field. It can be noted that most interest in chatbot usability is on *effectiveness* and *satisfaction*.

After conducting the SMS and analyzing the literature with respect to the usability of chatbots, the primary studies were classified from four different perspectives: usability techniques, usability characteristics, research methods and type of chatbots. These categories are reviewed next.

A. Usability Techniques

The primary studies in this category identify the adoption of usability techniques from HCI. This is the second-most studied group in the literature. The usability techniques are shown in Table I. From the analysis of the papers, we found that *questionnaires* and *interviews* are most commonly used.

TABLE I. USABILITY TECHNIQUES

Usability Techniques	Primary studies
Questionnaire (SUS/ad-hoc)	[1][3][13][14][15][16][17][18][19][20][21][22][23][24][25][26]
Interview	[15][21][22][23][24][26][27][28][29]
Think-aloud	[15][19][22]
Direct observation	[20]
Cognitive walkthrough	[22]

In most cases, two or more techniques are combined for the usability evaluation. Each of these methods has its own characteristics, and cannot fully meet all requirements of the usability test in isolation. Hence, it is necessary to combine various methods. For example, in [20], direct observation and the *System Usability Scale* (SUS) questionnaire are jointly used. In [21], questionnaires and interviews are used together in every research phase. In [22] the authors conduct a usability test to compare the usability of three chatbot platforms by using a SUS questionnaire, think-aloud and interview to rate the feedback from participants. A post-task questionnaire and an open-ended interview were used together in [26]. In [15], they video-recorded the experiment process for a retrospective think-aloud, and then conducted an interview and a questionnaire after accomplishing the tasks. In [24], user testing is combined with questionnaires and interview. In [23], they conducted semi-structured interviews and used different standard questionnaires together in the first and last evaluation period. In the middle period, to gather more comprehensive information, they used the SUS questionnaire. In [13], they developed a pre-study questionnaire to illustrate the types of interactions perceived to be the most frequent with an Alexa chatbot. In [1], participants were asked to fill the metrics to measure their user experience, and also were asked to compare two different interfaces and justify their responses.

In some cases, the authors used just one single technique to measure usability. In [14], they conducted a survey using a questionnaire. In [16], though they mainly used questionnaires to measure usability, they track user experience through different questionnaires from different periods with open questions. In [27], they used structured interviews. In [17], different questionnaires were used in three different periods of

the experiment. In [18], the authors used questionnaires to measure quantitative and qualitative evaluations of the new NLP method used by the chatbot. In [29], during the interview, participants had to explain the difficulties they had. In [28] to avoid excessive verbosity and to use verbal instead of text feedback, they used interview with open questions. In [25], to evaluate their web client, the authors used a questionnaire related to reliability, usability, and functionality of the system. Overall, we can conclude that the technique used depends on the specific conditions, while there is no standard proposal.

B. Usability Characteristics

According to the primary studies, usability characteristics are mainly identified in three aspects: Effectiveness, Efficiency and Satisfaction.

1) *Effectiveness*: Effectiveness is defined as the accuracy and completeness with which users achieve specified goals in HCI [30][31]. From Figure 1, most papers consider effectiveness as an essential factor when evaluating the usability of chatbots. Table II shows more details on the used effectiveness criteria.

TABLE II. EFFECTIVENESS

Measures of Effectiveness	Primary studies
Task completion	[1][13][22][25][27]
Accuracy	[17][18][24][25][26][28]
Recall	[18][25]
Experts and Users' assessment	[14][15][21][24]

We have identified task completion, accuracy of chatbot reply, comparison with recall and expert assessment as the main means to assess effectiveness. In [14] by gathering feedback from experts and potential users, they evaluate grading of the perceived quality of effectiveness of the chatbot [31] and find some shortcomings and possible solutions that will enhance the application's usability for its intended audience. In these works, the number of correct responses or interventions indicates the accuracy (to measure if user achieve specified goal [30]) and recall (users' ability to recall information from the interface [31]). The result shows that most chatbots achieve the required accuracy and recall of response [25][26]. For example, through comparing with other chatbots with similar functionality for completing the task, the authors in [1] proved their e-commerce chatbot performs better than the default chatbot. In [18], according to the result of the questionnaire, 80% participants claimed that the content of the retrieved information is clear and useful. In [27], the authors measure the number of users who complete the task (interview) through two different tools, showing that the chatbot has higher acceptability. To identify the measures of characteristics accuracy and recall, the works [30][31] have been followed.

However, there are still problems to achieve a high level of task completion and accuracy of the chatbot reply. In [13], 19 incomplete tasks were reported, because of an ill-defined system design. In [17], during the evaluation, there were some problems with the DBpedia semantic entry point, which

affected the accuracy of some of the users. In [28], 46 entries were negotiated, of which 7 (15.2 %) did not correspond correctly to the user's original wishes, but when a participant use more lengthy sentences, he produced noticeably more utterances compared to the average of the others. This problem mainly resulted from the inability of the system to process long, convoluted utterances properly and lacking the ability to guide the user during the interaction. In [24], the chatbot generated unnecessary information in response to highly structured conversations. In [25], the factor affecting the accuracy of chatbot reply is the need to handle one or more user conversation turns before providing the answer.

2) *Efficiency*: Efficiency relates to the resources expended in relation to the accuracy and completeness with which the users achieve their goals [30][31]. Most papers discuss *task completion time*, *mental effort* and *communication effort* to use the chatbot, as shown in Table III.

TABLE III. EFFICIENCY

Measures of Efficiency	Primary studies
Task completion time	[1][29]
Mental effort	[1][3]
Communication effort	[17][20][21]

In [1], the authors compare the number of views and average time the participants took in completing a task with the Convey chatbot, and a default one. The results showed they spent more effort and time with Convey in performing the task.

Perceived autonomy and competence are factors favoring efficiency in chatbot usability [3]. In [17], it was noted that, since the chatbot can correct erroneous inputs, users do not need to spend much communication effort when talking to the chatbot. In addition, less communication effort makes the chatbot easier to operate. In [20] the authors count the number of participants' cumulative assertions to measure the communication effort. Its steady increase demonstrates that users can use the chatbot efficiently in short time. Finally, users spend more communication effort when the chatbot has limited conversational ability, as discussed in [21].

3) *Satisfaction*: This is the largest group of papers within the primary studies. Satisfaction is defined as the degree to which user needs are satisfied when a product or system is used in a specified context of use [30][31]. The measures of satisfaction include *ease-of-use*, *context-dependent questions*, *satisfaction before* and *during use*, *complexity control*, *physical discomfort* of the interface, *pleasure*, the *willing of use* the chatbot *again*, and enjoyment and *learnability*. From Table IV, the ease-of-use, willing to use the chatbot again and user experience are the main measures of satisfaction used. Emotional aspects such as perceived utility, pleasure, comfort, are also considered in [24], and are related to the user experience. Among the primary studies, works have been found measuring the user experience mainly considering the physical discomfort and pleasure. These works are highlighted with a rectangle in the Table IV.

It must be noted that chatbots have more exploration space for interaction with users. A physical chatbot was proposed in

[16] to support self-management of diabetes by children. The usability evaluation included capabilities, social presence, and the quantity of speech and movements. Children stated that physical chatbots were more (inter)active, more present and capable of doing different things, such as dancing. Chatbots with actual images or entities are more likely to establish relationships with users, improving their experience. In [29], a combination of speech-and-gesture makes users get well better with the chatbot. In [22], the authors compared Pandorobot with two other chatbots. Overall, Pandorobot's voice sounded less robotic, entertaining users better.

TABLE IV. SATISFACTION

Measures of Satisfaction	Primary studies
Ease-of-use	[1][14][16][17][18][19][22][23][24][26]
Context-dependent question	[22][23][29]
Before use	[14][21][23]
During use	[3][21]
Complexity control	[14][18][23][25]
Physical discomfort	[14][18][21]
Pleasure	[1][13][16][17][21][22][26][29]
Want to use again	[1][14][16][17][21]
Learnability	[15][16][22]

Besides, more flexibility and speech commands context-dependent are required for better usability. In [1] participants mentioned that a shopping chatbot was easy to use since it tracked their search history. In [17] some users did not consider they needed an affective enhanced semantic chatbot at home. In [23] the authors observed that the acceptance of the chatbot decreases since its response mismatched the users' initial expectations. Potential explanations for such inconsistencies might include fundamental differences in user expectations for the chatbot and the emphasis on the interactive and entertaining qualities of the system over its informational value. The user background should be considered a key point in evaluating satisfaction. Cultural, socio-economical and personal preferences can influence the opinions towards the chatbot. In [23], the authors noticed that users in the Netherlands were more experienced with technology than in the other two countries of the study, therefore their expectations towards the novel technology were higher. In [15] users with higher technical knowledge learned quicker to use the chatbot.

C. Research Methods

The research methods used by the authors of the primary studies within this group include surveys of chatbots users' experience, experiments of using chatbots to realize some given tasks, usability tests, case studies and quasi-experiments. The research methods are detailed in Table V. The most common research methods include *survey*, *experiment* and *usability test*. In most experiments, very simple tasks are proposed. For example, using Apple Siri to find an inexpensive hotel in Osaka [15], search a flight ticket and hotel [3], whether

a simple chatbot can be appropriately used as a delivery mechanism [24], buying shoes [1], measuring the quality and quantity of the information retrieved [18], taking a structured interview with chatbot [27], or playing a game [29]. However, real-life situations are more complicated.

TABLE V. RESEARCH METHODS

Research methods	Primary studies
Survey	[13][16][19][22][24][25][26]
Experiment	[1][3][13][14][15][16][17][19]
Usability test	[18][19][22][25][27][28]
Case study	[14][20][23][29]
Quasi-experiment	[20][21][25]

Rather than aiming to fully recreate the real-world task, simulation-based assessment should incorporate psychologically relevant aspects and situations from the real-world task and the environment, such as time-pressure, or high uncertainty. In [17], the authors show that when the chatbot has visual appearance and emotions, users do not notice small changes in voice and facial expressions. The experiment concludes that it is not necessary to use extremely accurate facial expressions for realistic use. In [21], the authors deployed a digital pet avatar in the participants' home for 3 months to simulate real-life situations as much as possible. In [20], experiments were designed in a complex way, to simulate real-world situations. This is sometimes necessary, because if we were unable to use the chatbot effectively with a design as realistic as possible (but nonetheless simplified), it would be unlikely to be effective under more challenging conditions for the military, law enforcement and others in safety-critical real-world environments.

In most cases, researchers make comparisons. For example, in [14], the authors compare a chatbot with a similar one or with a similar application. Research methods can also be combined, which typically yields better results. In [13], a pre-survey questionnaire was performed to assess the usability of an Alexa chatbot. Then, an experiment was conducted to investigate specific problems. Machine learning (ML) algorithms, in combination with cloud-based databases can be used to solve some current shortcomings of handling natural language (e.g., chatbots can't recognize the words that they haven't been programmed for, and some chatbots speak unnatural language [24][26]).

D. Type of Chatbots

The AIML technology is still widely used in the design of chatbots [24][25]. However, the use of chatbots using NLP is growing [20]. For example, the PAL project [16] can generate reasonable feedback through user-entered information. In [13], the authors show that the chatbot can be used via natural language phonic control, to perform search, entertainment, and to control other devices. In [14], more users are satisfied with the chatbot, due to its speaker functionality and natural conversation flow. In [18], the authors used Object Relational Mapping (ORM) frameworks to improve the process of

generating SQL statements from NL queries. Many chatbots are built as Embodied Conversational Agents (ECA), and there are increasing number of chatbots with image, sound and personality [17][21][23][29]. However, sometimes chatbots have negative emotions. When the ECA has a negative personality, it tends to ignore or blame the user [17]. In addition, the chatbot is required to have the ability to learn and adapt to its user context to be useful [23]. Therefore, complete evaluations should be carried out to obtain a better comprehension of these issues.

V. DISCUSSION AND VALIDITY THREATS

The analysis reveals that the incorporation of usability techniques in the chatbot development process in a formalized manner is not strongly reflected in the primary studies. We found three papers reviewing the chatbot literature: one discussing the conversational interfaces, patterns, and paradigms [9], one investigating design techniques for conversational agents [10], and a systematic review of conversational agents in healthcare [11]. None of them does a SMS in chatbots usability, which proves our work is original. Judging by the increase in publications since 2015, the integration of usability of chatbots is of notable interest. However, there is no agreement on what would be a formalized and more systematic integration yet. Therefore, it is an open problem that requires more research effort. Even though the literature retrieved by the SMS provides a picture of chatbot usability, no paper provides generally applicable guidelines for chatbots usability. On one hand, the validity of our SMS is threatened by including only papers written in English. On the other, the authors of an SMS may make errors of judgement when analyzing the relevant publications. This is a horizontal rather than a vertical analysis, on which ground relevant publications may have been overlooked. Additionally, although the terms used in the search string were the most commonly accepted by other authors, other terms used describing relevant publications may have been overlooked. Finally, the publications were evaluated and classified based on the judgment and experience of the authors, and other researchers may have evaluated the publications differently.

VI. CONCLUSIONS

This paper has described an SMS study conducted to answer two research questions: **RQ1.** *What is the state of the art of usability in the development of chatbots?* We retrieved 19 primary studies dealing with integration from four different perspectives: usability techniques, usability characteristics, research methods and types of chatbots. The usability techniques are applied to evaluate the usability of the developed chatbot, but not in the analysis and design activities of the chatbot. The procedure more frequently followed to evaluate the usability of chatbot is to select a group of subjects to use the chatbot freely or perform certain tasks and then measure satisfaction with a SUS survey.

RQ2. *How to evaluate the usability of chatbots using HCI principles?* The evaluation of the usability of chatbots must be done considering the context of use, i.e. the environment where the chatbot will be used, and with representative subjects to whom the chatbot is directed. The most commonly used

methods are surveys, experiments and usability tests. The experimentation and replication of experiments is key within HCI. Achieving successful replicas in a discipline allows its results to be added to previous ones, making the body of knowledge grow. However, there is an absence of controlled experiments and replicas measuring chatbots usability.

There are many ways for practitioners to apply the usability material in this paper: (i) The chatbot implementation team can use usability characteristics (Tables II-IV) as checklists to help them solve critical problems, and (ii) comparing the test results of the same system at different times can check whether the usability characteristics is improved or decreased.

The real-life application of a chatbot will save time to companies, leading to financial gain because of the tasks it is able to take on. As the intelligence and technology of chatbots evolve, they will take on more responsibilities. The chatbot industry is very much interested in the adoption of usability techniques in its development process. On this ground, there is a need for usability-aware design guidelines.

ACKNOWLEDGMENT

Work supported by the R&D program of the Madrid Region (S2018/TCS-4314), and the Spanish Ministry of Science (project MASSIVE, RTI2018-095222-B-I00).

REFERENCES

- [1] M. Jain, R. Kota, P. Kumar and S.N. Patel. "Convey: Exploring the use of a context view for chatbots". *Proc. CHI Conference on Human Factors in Computing Systems*, p. 468. 2018.
- [2] ISO 9241-11. "Ergonomic requirements for office work with visual display terminals (VDTs)—Part II guidance on usability". 1998.
- [3] Q.N. Nguyen and A. Sidorova. "Understanding user interactions with a chatbot: A self-determination theory approach". In *Americas Conference on Information Systems 2018: Digital Disruption*. 2018.
- [4] K.Panetta. "Gartner's top 10 strategic technology trends for 2017". Online: <https://www.gartner.com/smarterwithgartner/gartner-top-10-technology-trends-2017/>. 2016. [Accessed march 26-2019]
- [5] J. Pereira and O. Díaz. "A quality analysis of facebook messenger's most popular chatbots". *Proc. of ACM SAC*, pp. 2144-2150. 2018.
- [6] C. Messina. "2016 Will be the year of conversational commerce". Online: <https://medium.com/chris-messina/2016-will-be-the-year-of-conversational-commerce-1586e85e3991>. 2016. [Accessed 12/14/2018].
- [7] J. Lester, K. Branting and B. Mott. "Conversational agents". *The Practical Handbook of Internet Computing*, Chapman and Hall/CRC, pp. 220-240. 2004.
- [8] L. Sullivan. "Facebook chatbots hit 70% failure rate as consumers warm up to the tech". Online: <https://www.mediapost.com/publications/article/295718/>. 2017. [Accessed march 26-2019]
- [9] L.C. Klopfenstein, S. Delpriori, S. Malatini and A. Bogliolo. "The rise of bots: A survey of conversational interfaces, patterns, and paradigms". *Proc. Conference on Designing Interactive Systems*, pp. 555-565. 2017.
- [10] K. Ramesh, S. Ravishankaran, A. Joshi and K. Chandrasekaran. "May. A survey of design techniques for conversational agents". In *Int. Conf. on Inform., Commun. and Computing Technology*, pp. 336-350. 2017.
- [11] L. Laranjo, A.G. Dunn, H.L.Tong, A.B.Kocaballi, J.Chen, R.Bashir, D.Surian, B.Gallego, F.Magrabi, A.Y. Lau and E.Coiera. "Conversational agents in healthcare: a systematic review". *Journal of the American Medical Informatics Associat.*, 25(9), pp.1248-1258. 2018.
- [12] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. "Systematic mapping studies in software engineering". In *Proceedings of the 12th Int. Conf. on Evaluation and Assessment in Soft. Eng.*, pp. 71-80. 2008.
- [13] I. Lopatovska, K. Rink, I. Knight, K. Raines, K. Cosenza, H. Williams, P. Sorsche, D. Hirsch, Q. Li, and A. Martinez. "Talk to me: Exploring user interactions with the amazon alexa". *Journal of Librarianship and Information Science*. 2018.
- [14] A. Cheng, V. Raghavaraju, J. Kanugo, Y.P. Handrianto and Y. Shang. "Development and evaluation of a healthy coping voice interface application using the Google home for elderly patients with type 2 diabetes". In *Consumer Communications & Networking Conference (CCNC)*, 2018 15th IEEE Annual, pp. 1-5. 2018.
- [15] M.L. Chen and H.C. Wang. "How personal experience and technical knowledge affect using conversational agents". *Proc. 23rd Inter. Conf. on Intelligent User Interfaces Companion*, p. 53. 2018.
- [16] C. Sinoo, S. van der Pal, O.A.B. Henkemans, A. Keizer, B.P. Bierman, R. Looije and M.A. Neerincx. "Friendship with a robot: Children's perception of similarity between a robot's physical and virtual embodiment that supports diabetes self-management". *Patient education and counseling*. 2018.
- [17] J. Pérez, Y. Sánchez, F.J. Serón, and E. Cerezo. "Interacting with a semantic affective ECA". *Proc. Conf. Intelligent Virtual Agents*, pp. 374-384. 2017.
- [18] A. Alghamdi, M. Owda and K. Crockett. "Natural language interface to relational database (NLI-RDB) through ORM". *Advances in Intelligent Systems and Computing*, vol 513. Springer, Cham 2017.
- [19] M.L. Tielman, M.A. Neerincx, R. Bidarra, B. Kybartas and W.P. Brinkman. "A therapy system for post-traumatic stress disorder using a virtual agent and virtual storytelling to reconstruct traumatic memories". *Journal of medical systems*, 41(8), p.125. 2017.
- [20] A. Preece, W. Webberley, D. Braines, E.G. Zaroukian and J.Z. Bakdash. "SHERLOCK: Experimental evaluation of a conversational agent for mobile information tasks". *IEEE Transactions on Human-Machine Systems*, 47(6), pp.1017-1028. 2017.
- [21] N.C. Chi, O. Sparks, S.Y. Lin, A. Lazar, H.J. Thompson and G. Demiris. "Pilot testing a digital pet avatar for older adults". *Geriatric Nursing*, 38(6), pp.542-547. 2017.
- [22] J. Saenz, W. Burgess, E. Gustitis, A. Mena and F. Sasangohar. "The usability analysis of chatbot technologies for internal personnel communications". In *67th Annual Conference and Expo of the Institute of Industrial Engineers*, pp. 1357-1362. 2017.
- [23] C. Tsiouri, J. Quintas, M. Ben-Moussa, S. Hanke, N.A. Nijdam and D. Konstantas. "The CaMeLi Framework—A multimodal virtual companion for older adults". In *Proceedings of SAI Intelligent Systems Conference*, pp. 196-217. 2018.
- [24] D. Elmasri and A. Maeder. "A conversational agent for an online mental health intervention". *Proc. Int. Conf. Brain and Health Informatics*, pp. 243-251. 2016.
- [25] A.I. Niculescu, K.H. Yeo, L.F. D'Haro, S. Kim, R. Jiang and R.E. Banchs. "Design and evaluation of a conversational agent for the touristic domain". In *APSIPA*, pp. 1-10. 2014.
- [26] S. Tegos, S. Demetriadis and T. Tsiatsos. "A configurable conversational agent to trigger students' productive dialogue: a pilot study in the CALL domain". *International Journal of Artificial Intelligence in Education*, 24(1), pp.62-91. 2014.
- [27] J.A. Micoulaud, P. Sagaspe, E. De Sevin, S. Bioulac, A. Sauteraud and P. Philip. "Acceptability of embodied conversational agent in a health care context". *Proc. Conf. Intelligent Virtual Agents*, pp. 416-419. 2016.
- [28] R. Yaghoubzadeh, K. Pitsch and S. Kopp. "Adaptive grounding and dialogue management for autonomous conversational assistants for elderly users". *Proc. Conf. Intelligent Virtual Agents*, pp. 28-38. 2015.
- [29] D. Novick and L.M. Rodríguez. "Extending empirical analysis of usability and playability to multimodal computer games". In *Intern. Conf. of Design, User Experience, and Usability*, pp. 469-478. 2016.
- [30] ISO/IEC 25010. "Systems and software engineering—systems and software quality requirements and evaluation (SQuARE)—system and software quality models". 2010.
- [31] K. Hornbæk. "Current practice in measuring usability: Challenges to usability studies and research". *International Journal of Human-Computer Studies*, 64(2), pp. 79-102. 2006.

Extending Behavior-Driven Development for Assessing User Interface Design Artifacts

Thiago Rocha Silva

Department of Computer Science,
Norwegian University of Science and
Technology (NTNU), Norway
thiago.silva@ntnu.no

Marco Winckler

SPARKS-i3S,
Université Nice Sophia Antipolis
(Polytech), France
winckler@unice.fr

Hallvard Trætteberg

Department of Computer Science,
Norwegian University of Science and
Technology (NTNU), Norway
hal@ntnu.no

Abstract — This paper presents a scenario-based approach to specify requirements and tests by extending Behavior-Driven Development (BDD) with the aim of ensuring the consistency between user requirements and user interface design artifacts. The approach has been evaluated by exploiting user requirements specified by a group of potential Product Owners (POs) for a web system to book business trips. Such requirements gave rise to a set of User Stories that have been refined and used to automatically check the consistency of task models, user interface (UI) prototypes, and final UIs of the system. The results have shown our approach was able to identify different types of inconsistencies in the set of analyzed artifacts and consistently keep the semantic traces between them.

Index Terms — Behavior-Driven Development (BDD); User Interface Design Artifacts; Automated Requirements Assessment.

I. INTRODUCTION

Modeling is recognized as a crucial activity to manage the abstraction and the inherent complexity of developing software systems. As a consequence, software systems tend to be designed based on several requirements artifacts which model different aspects and different points of view about the system. Considering that different phases of development require distinct information, resultant artifacts from modeling tend to be very diverse throughout the development, and ensuring their consistency is quite challenging [1]. To face this challenge, extra effort should be put on getting requirements described in a consistent way across the multiple artifacts. Requirements specifications should not, for example, describe a given requirement in a user interface (UI) prototype which is conflicting with its representation in a task model.

Behavior-Driven Development (BDD) [2] has aroused interest from both academic and industrial communities as a method allowing specifying testable user requirements in natural language using a single textual artifact. BDD describes User Stories (US) [3] and scenarios in a easily understandable way for both technical and non-technical stakeholders. In addition, BDD scenarios allow specifying “executable requirements”, i.e. requirements that can be directly tested from their textual specification. Despite providing support to automated testing of user requirements, BDD and other testing approaches essentially focus on assessing fully interactive artifacts such as full-fledged (final) versions of user interfaces. Automated assessment of model-based artifacts such as task models, UI prototypes, etc. is not supported.

Motivated by such a gap, we have researched and developed an approach based on BDD and User Stories to support the specification and the automated assessment of functional aspects of user requirements on user interface design artifacts such as task models, UI prototypes, and final UIs [4]–[8]. This paper presents a refined version of this approach and summarizes the new results we got in a case study exploiting User Stories specified by potential Product Owners (POs) to automatically assess user interface design artifacts for a web system to book business trips. The following sections present the foundations of this work as well as the refined version of our approach and a brief discussion of the results obtained with this case study.

II. FOUNDATIONS

A. Behavior-Driven Development (BDD)

According to Smart [9], BDD is a set of software engineering practices designed to help teams focus their efforts on identifying, understanding, and building valuable features that matter to businesses. BDD practitioners use conversations around concrete examples of system behavior to help understand how features will provide value to the business. BDD encourages business analysts, software developers, and testers to collaborate more closely by enabling them to express requirements in a more testable way, in a form that both the development team and business stakeholders can easily understand. BDD tools can help turn these requirements into automated tests that help guide the developer, verify the feature, and document the application.

BDD specification is based on User Stories and scenarios which allow to specify executable requirements and test specifications by means of a Domain-Specific Language (DSL) provided by Gherkin. User Stories were firstly proposed by Cohn [3]. North [10] has proposed a particular template to specify them in BDD and named it as “BDD story”:

Title (one line describing the story) Narrative: As a [role], I want [feature], So that [benefit] Scenario 1: Title Given [context], When [event], Then [outcome]
--

In this template, BDD stories are described with a *title*, a *narrative* and a set of *scenarios* representing the acceptance criteria. The *title* provides a general description of the story, referring to a feature this story represents. The *narrative* describes the referred feature in terms of the role that will benefit from the feature (“*As a*”), the feature itself (“*I want*”), and the

benefit it will bring to the business (“*So that*”). The acceptance criteria are defined through a set of *scenarios*, each one with a title and three main clauses: “*Given*” to provide the context in which the scenario will be actioned, “*When*” to describe events that will trigger the scenario and “*Then*” to present outcomes that might be checked to verify the proper behavior of the system. Each one of these clauses can include an “*And*” statement to provide multiple contexts, events and/or outcomes. Each statement in this representation is called a *step*.

B. User Interface Design Artifacts

1) *Task Models*: Task models provide a goal-oriented description of interactive systems but avoiding the need of detail required for a full description of the user interface. Each task can be specified at various abstraction levels, describing an activity that has to be carried out to fulfill the user’s goals. By modeling tasks, designers are able to describe activities in a fine granularity, for example, covering the temporal sequence of tasks to be carried out by the user or system, as well as any preconditions for each task [11]. The use of task models serves multiple purposes, such as better understanding the application under development, being a “record” of multidisciplinary discussions between multiple stakeholders, helping the design, the usability evaluation, the performance evaluation, and the user when performing the tasks. Task models are also useful as documentation of requirements both related with content and structure. HAMSTERS [12] is a tool-supported graphical task modeling notation for task modeling. In HAMSTERS, tasks can be of several types such as abstract, system, user, and interactive tasks. Temporal relationships between tasks are represented by means of operators. Operators can also be of several types such as enable, concurrent, choice, and order independent operators. The temporal operators allow extracting usage scenarios for the system. This is done by following the multiple achievable paths in the model, with each combination of them generating an executable scenario that can be performed in the system.

2) *User Interface (UI) Prototypes and Final UIs*: A UI prototype is an early representation of an interactive system. They encourage communication, helping designers, engineers, managers, software developers, customers and users to discuss design options and interact with each other. Prototypes are often used in an iterative design process where they are refined and become more and more close to the final UI through the identification of user needs and constraints. While the beginning of the project requires a low-level of formality with UI prototypes being hand-sketched in order to explore design solutions and clarify user requirements, the development phase requires more refined versions frequently describing presentation and dialog aspects of the interaction. By running simulations on prototypes, we can determine and evaluate potential scenarios that users can perform in the system [13]. The presentation aspect of full-fledged user interfaces frequently corresponds to how the user “see” the system. From the user’s point of view, the presentation of a user interface actually is the system, so if some feature is not available there, then it does not exist at all. Mature UI versions are the source for acceptance testing and will be used by users and other stakeholders to assert whether or not features can be considered as done.

III. THE PROPOSED APPROACH

Our proposed approach for assessing the considered artifacts is illustrated in Figure 1, where User Story scenarios are used to ensure consistency in our target artifacts (task models, UI prototypes and final UIs). Therein are exemplified five steps of scenarios being tested against equivalent tasks in task model scenarios, and the equivalent interaction elements in UI prototypes and final UIs. In the first example, the step “*When I select ‘<field>’*” corresponds to the task “*Select <field>*” in the task model scenario. Such a correspondence is due to the fact that the step and the task represent the same behavior, i.e. selecting something, and both of them are placed at the first position in their respective scenario artifacts. The interaction element “*field*” that will be affected by such a behavior will be assessed on the UI prototype and on the final UI. In both artifacts, such a field has been designed with a *CheckBox* as interaction element. The semantics of the interaction in *CheckBoxes* is compatible with selections, i.e. we are able to select *CheckBoxes*, so the consistency is assured.

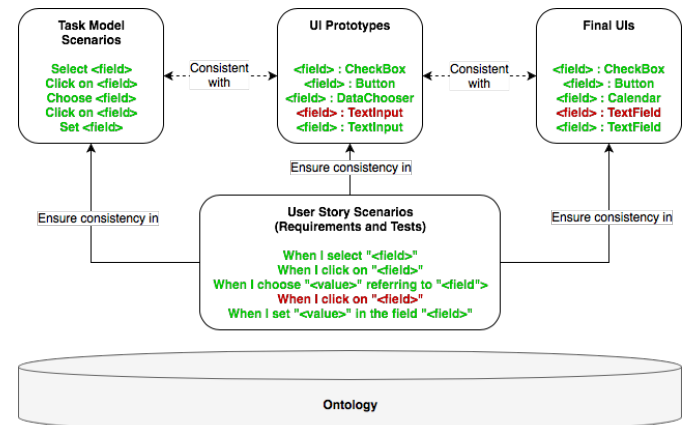


Figure 1. The approach for assessing the different UI artifacts.

The same is true in the example with the second step (“*When I click on ‘<field>’*”). There is a corresponding task “*Click on <field>*” at the same second position in the task model scenario, and the interaction element “*Button*”, that has been chosen to address this behavior in both the UI prototype and the final UI, is semantically compatible with the action of clicking, thus the consistency is assured as well. In the third example, the step “*When I choose ‘value’ referring to ‘field’*” is also compatible with the task “*Choose <field>*” in the task model, and with the interaction elements *DataChooser* and *Calendar*, respectively in the UI prototype and in the final UI. Notice that, despite being two different interaction elements, *DataChooser* and *Calendar* support a similar behavior, i.e. both of them support the behavior of choosing values referring to a field.

The example provided with the fourth step (“*When I click on ‘<field>’*”) illustrates an inconsistency being identified. Even though there exists a corresponding task in the task model scenario, the interaction elements that have been chosen to address this behavior (*TextInput* in the UI prototype and *TextField* in the final UI) are not compatible with the action of clicking, i.e. such kind of interaction element does not semantically support such an action. The semantics of *TextInputs* (or *TextFields*) is receiving values, not being clicked. Such an example is provided with the fifth step (“*When I set*

'value' in the field '<field>'''). For this step, the consistency is assured because *TextInputs* and *TextFields* support the behavior of having values being set on them. All this semantic analysis is supported by the use of an ontology that models the interaction elements and the interactive behaviors they support [14], [15].

The present strategy for assessment allows tracking some key elements in the UI design artifacts and check whether they are consistent with the user requirements. The solution has been implemented in Java integrating multiple frameworks such as JBehave, JDOM, JUnit, and Selenium WebDriver.

A. Alternatives for Performing the Approach

Depending on the project phase, our approach can be applied in two ways. The first one is applied when the project is running, and artifacts have already been designed. In such a case, our approach can be used to assess such artifacts, indicating where they are not in accordance with the specified requirements. The second one refers to a project in the beginning, where no artifacts have been designed yet. In this case, by using the ontology, they can be modeled in a consistent way from the beginning, taking into account the possible interactions supported by each interaction element on the UI.

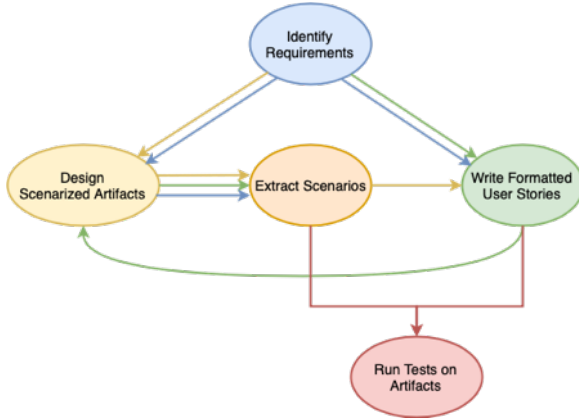


Figure 2. The graph of options for performing our approach (colors are used to visually identify the different paths).

Figure 2 illustrates the resultant graph of options considered. The colored lines indicate the possible paths to be taken in the workflow. The yellow path indicates the design of scenarized artifacts before writing formatted User Stories. The green path indicates the opposite, while the blue path indicates both activities in parallel. Notice that regardless the path chosen, the extraction of scenarios is only possible after having designed the scenarized artifacts, and the identification of requirements is a precondition for all the other activities. Finally, to run tests on the artifacts, it is required to have extracted scenarios and written the User Stories. The approach benefits from the independence for testing artifacts, i.e. tests can run on a single artifact or on a set of scenarized artifacts which will be targeted at a given time.

IV. CASE STUDY

To investigate the potential of the approach, we have conducted a case study with an existing web system for booking business trips. We have studied the current implementation of user requirements in this system, and by applying a manual reverse engineering, we redesigned the appropriate task models

and UI prototypes for the system. Based on a set of User Stories collected in a previous study [16], we refined it to simulate the assessment of the resultant user interface design artifacts. The aim of this present study is to provide a preliminary evaluation regarding the extent of inconsistencies our approach is able to identify in the targeted artifacts.

We started the study by setting up an initial version of User Stories before reengineering initial versions of task models (in HAMSTERS) and UI prototypes (in Balsamiq) from the existing web system. After getting a first version of task models, we extracted a representative set of scenarios from them. By following our strategy for testing, we parsed and ran the initial version of User Stories against the initial set of extracted scenarios. As the strategy we follow for testing scenarios in task models parses all the steps of each scenario at once, the first round of results was obtained with a single battery of tests. Following this step, we ran the same initial version of User Stories against initial versions of Balsamiq prototypes. Unlike the strategy for testing task models, the strategy we follow for testing UI prototypes and final UIs parses each step of each scenario at a time, so if an error is found out, the test stops until the error is fixed. That requires to run several batteries of tests until having the entire set of scenarios tested. Consequently, at the end of running, the tested scenarios are fully consistent with the UIs. Finally, we analyzed the testing results and the main types of inconsistencies identified in each artifact.

In total, we set up for assessment 3 User Stories with 15 different scenarios, reengineered 3 task models (and extracted 10 scenarios from them), reengineered 11 UI prototypes, and tested 7 different final UIs. For scenarios extracted from task models, testing results return the equivalent position of each task in the US scenarios. For UI prototypes and final UIs, the expected result for each step is the presence on the UI of one and only one of the supported interaction elements designed to address a given interactive behavior.

TABLE I. RESULTS AFTER ASSESSING THE ARTIFACTS

Artifact	Total (Steps Analyzed)	Results	
		Consistent	Inconsistent
Task Models	147	5	142
UI Prototypes	36	21	15
Final UIs	288	276	12

Table 1 summarizes the results obtained. For task models, the most common source of the 142 inconsistencies identified concerned the task gaps present in the beginning of the scenario. As the assessment is performed in the extracted scenarios which represent a sequential instance of the tasks in the task model, a task gap in the beginning causes a domino effect in the forthcoming tasks in the scenario. So even if the remaining tasks in the scenario are semantically equivalent to the respective steps, they will be shown as inconsistent once they will be found in wrong positions due to this gap. For UI prototypes, from the 15 inconsistencies identified, we noticed they were mainly due to interaction elements specified with different names in the step and in the prototype. For final UIs, the high number of consistent steps (276 out of 288) in the set of scenarios analyzed is due to the need of fixing the inconsistency found before moving forward to the next steps. This makes that the scenarios which

call previous ones, in order to reuse steps and reach a given state of the system, already have these steps fully consistent during the test. Most part of the inconsistencies on final UIs was due to interaction elements that do not carry a unique and single identifier (or carry a dynamically generated one) and, as such, cannot be reached during the test.

We could also remark that some of the inconsistencies identified showed to be more critical than others. While simple inconsistencies such as differences in names of tasks and fields, conflicts between expected and actual elements, and messages and elements not found are easy to solve, conflicts between specification and modeling, and different specification strategies for task models represent more critical problems. On UI prototypes, the presence of semantically inconsistent elements as well as more than one element to represent the same field are also critical problems. On final UIs, fields already filled-in denotes inconsistencies that exposes important design errors. During the test, we also noticed that some inconsistencies were due to a wrong specification of the step in the US scenario, and not to a problem in the design of the artifact itself. So, to fix these inconsistencies, steps of US scenarios needed to be modified during the battery of tests to obtain a consistent specification of user requirements and artifacts. An immediate consequence of this fact is that scenarios used to test a given version of an artifact may be different than the ones which were used to test another artifact previously. This makes regression tests essential to ensure that a given modification in the set of US scenarios did not break the consistency of other artifacts and ended up making some artifact (that so far was consistent with the requirements) inconsistent again.

As limitations of the approach, it is worthwhile to mention that its current version covers only the assessment of HAMSTERS task models, Balsamiq UI prototypes and web final UIs. The need of extracting scenarios from task models to perform testing in such artifacts, and tools that do not support yet the automatic classification of errors are other limitations.

V. CONCLUSION AND FUTURE WORKS

This paper summarizes the new results we got by applying our approach for specifying and checking the consistency of user requirements on core user interface design artifacts. Compared to plain-vanilla BDD, this approach benefits from (i) an extension to assess other software artifacts than final UIs, and (ii) a common vocabulary to be reused for specifying interactive scenarios without requiring developers to implement the mentioned behaviors. Compared to other approaches for assessing requirements and artifacts, the term “test” is usually not employed under the argument that such artifacts cannot be “run”, i.e. executed for testing purposes, so in practice they are just manually reviewed or inspected in a process called verification. Manual verification of the software outcomes is highly time-consuming, error-prone and even impracticable for large software systems. Fully interactive artifacts such as final UIs can in addition be validated by users who can interact with the artifact and assess whether its behavior is aligned with their actual needs. As within our approach we succeed automatically running User Stories on software artifacts for assessing their consistency with user requirements, we actually provide the “test” component for both verification and validation of artifacts in the software development. We consider this a big step towards

the automated testing (and not only the manual verification) of software artifacts by means of a consistent approach allowing fully verification, validation, and testing (VV&T).

Future works include evaluating the impact of maintaining and successively evolving the mentioned artifacts throughout a real software development process, besides investigating the suitability of the approach for assessing a wider group of artifacts, especially those related to conceptual aspects of software modeling such as class diagrams. Concerning the tools, the development of a plugin to suggest and autocomplete steps in the User Story scenarios based on the interactive behaviors of the ontology is also envisioned.

REFERENCES

- [1] M. Winckler and P. Palanque, “Models as Representations for Supporting the Development of e-Procedures,” in *Usability in Government Systems*, Elsevier, 2012, pp. 301–315.
- [2] D. Chelimsky, D. Astels, B. Helmkamp, D. North, Z. Dennis, and A. Hellesoy, *The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends*. Pragmatic Bookshelf, 2010.
- [3] M. Cohn, *User Stories Applied for Agile Software Development*. Addison-Wesley, 2004.
- [4] T. R. Silva and M. A. A. Winckler, “Towards Automated Requirements Checking Throughout Development Processes of Interactive Systems,” in *2nd Workshop on Continuous Requirements Engineering (CRE), REFSQ 2016*, 2016, pp. 1–2.
- [5] T. R. Silva, “Definition of a Behavior-Driven Model for Requirements Specification and Testing of Interactive Systems,” in *Proceedings of the 24th International Requirements Engineering Conference (RE 2016)*, 2016, pp. 444–449.
- [6] T. R. Silva, J.-L. Hak, and M. Winckler, “Testing Prototypes and Final User Interfaces Through an Ontological Perspective for Behavior-Driven Development,” in *HCSE 2016 and HESSD 2016*, LNCS, vol. 9856, Springer, 2016, pp. 86–107.
- [7] T. R. Silva, J.-L. Hak, and M. Winckler, “An Approach for Multi-Artifact Testing Through an Ontological Perspective for Behavior-Driven Development,” *Complex Systems Informatics and Modeling Quarterly*, no. 7, pp. 81–107, 2016.
- [8] T. R. Silva and M. Winckler, “A Scenario-Based Approach for Checking Consistency in User Interface Design Artifacts,” in *Proceedings of the 16th Brazilian Symposium on Human Factors in Computing Systems (IHC 2017)*, 2017, vol. 1, pp. 21–30.
- [9] J. F. Smart, *BDD in Action: Behavior-driven development for the whole software lifecycle*, 1 edition. Manning Publications, 2014.
- [10] D. North, “What’s in a Story?,” 2019. [Online]. Available: <https://dannorth.net/whats-in-a-story/>. [Accessed: 01-Jan-2019].
- [11] F. Paternò, C. Santoro, L. D. Spano, and D. Raggett, “W3C, MBUI - Task Models,” 2017. [Online]. Available: <http://www.w3.org/TR/task-models/>.
- [12] C. Martinie, P. Palanque, and M. A. Winckler, “Structuring and Composition Mechanisms to Address Scalability Issues in Task Models,” in *INTERACT 2011*, 2011, vol. 6948 LNCS, no. 3, pp. 589–609.
- [13] M. Beaudouin-Lafon and W. E. Mackay, “Prototyping Tools and Techniques,” in *Prototype Development and Tools*, 2000, pp. 1–41.
- [14] T. R. Silva, J.-L. Hak, and M. Winckler, “A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces,” *International Journal of Semantic Computing*, vol. 11, no. 04, pp. 513–539, 2017.
- [15] T. R. Silva, J.-L. Hak, and M. Winckler, “A Behavior-Based Ontology for Supporting Automated Assessment of Interactive Systems,” in *Proceedings of the 11th IEEE International Conference on Semantic Computing (ICSC 2017)*, 2017, pp. 250–257.
- [16] T. R. Silva, M. Winckler, and C. Bach, “Evaluating the usage of predefined interactive behaviors for writing user stories: an empirical study with potential product owners,” *Cognition, Technology & Work*, 2019.

A Method to Recommend Artifacts to New Tasks in Software Projects

Edson M. Lucas^{a,b}, Toacy C. Oliveira^a, Paulo S.C. Alencar^c

^aPESC/COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil

^bPolytechnic Institute (IPRJ/UERJ), Nova Friburgo, Brazil

^cDavid Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada

edmlucas@cos.ufrj.br, toacy@cos.ufrj.br, palencar@uwaterloo.ca

Abstract— The software development workflow typically involves developers executing tasks and manipulating artifacts. When developers receive a new task they typically envision a task context with the artifacts they intend to manipulate based on their past experiences. Given software projects may last several months, accumulating a vast amount of tasks, artifacts and developers, envisioning this initial task context may be difficult and error-prone. Developers have to walk-through months of past experiences or examine the experience of other developers, select similar tasks and then define the initial context. This paper introduces a method that helps developers defining the initial task context by combining interaction information over artifacts with text information of tasks. First, the Method uses the Clustering technique to organize project tasks into similar groups by interaction in artifacts. Then, the Method uses the Natural Language Processing technique to associate a new task with groups of similar tasks by interaction. The evaluation shows that the clustering of similar tasks by interaction produces similar tasks assigned with artifacts that will be edited by new tasks. The association of new tasks with similar groups by interaction indicates correlation between textual similarity and interaction similarity.

Keywords—component; interaction; task context; recommendation

I. INTRODUCTION

Software development projects last for months or years, where developers interact with each other and manipulate many artifacts. A typical task context in software development is formed by a set of artifacts that the developer uses to perform a task [1], [2]. Another widely explored context variable is the identification of task experts [3], as well as by experts in similar tasks to the new task [4].

When the developer receives a task to correct an error or add new functionality to the software, the developer usually starts by searching for artifacts that he needs to change to accomplish the task. This search is based on the developer's experience and the human capacity to remember of the artifacts already used in previous similar tasks [5]. In addition, the search is time-consuming, even when using the traditional navigation and

textual search tools available in development environments, e.g., Eclipse Project Explorer.

In this scenario, recommending an initial task context can help developers in performing new tasks. The recommendation tools in Software Engineering aim to reduce the uncertainties of the future through the analysis of project history, and in a more specific way, according to Robillard et al. [6], these tools are software applications that provides information items estimated as valuable to a software engineering task in a given context.

Aiming at inferring new task context, some works focus on the recommendation of artifacts, i.e., source code, files, classes, packages [1], [7]–[11], others in the recommendation of artifact experts [12], [13]. Proposals [14], [15] and [16] recommend experts to new tasks. Malheiros et al. [17] and Ashok et al. [18] recommend similar tasks to new task, while Wang et al. [4] and Ashok et al. [18] also recommend experts on similar tasks to new task.

This paper presents part of the Tacin method (Task Context based on Interactions) to recommend a new task context in software development projects. First, we defined that two tasks are similar by interaction if the developers edited at least one common artifact while performing the tasks. The weight of similarity is equal to the number of artifacts in common. After, the Method uses the Clustering technique to organize project tasks into similar groups by interaction in artifacts [19]. Then, the Method uses the Natural Language Processing technique to associate a new task with groups of similar tasks by interaction [20]. In this article, the edit interaction values are captured by Mylyn, an interaction model [28].

In the evaluation, the clustering of similar tasks by interaction indicated the production of task groups that had only 6% of the artifacts available in the project, but containing 77% of the artifacts that will be edited by new tasks. The association of new tasks with similar groups by interaction reduced by 90% the available artifacts and produced a recall of 52%. Therefore, the evaluation indicates success in Clustering and need for improvement in Natural Language Processing technique, i.e., improve the recall to a percentage closer to 77%.

This paper is organized as follows. Section II defines Clustering and Natural Language Processing. Section III presents the Method to recommend artifacts to new tasks in software projects. Section IV assesses the Method based on

average hits on recommending artifacts to new tasks. Section VI reviews related work and Section VI concludes our paper with a brief description of future work.

II. BACKGROUND

Developers interact with artifacts and collaborate among them to perform tasks in software development projects. The task objective is generally expressed by text. For example, the Mylyn Docs project uses a *short description* in natural language (English) to express an error. So, developer uses this textual information to search artifacts from the project's artifact database to correct it.

Table 1. Short description of task 245759 from the Mylyn Docs project.

	TaskId 245759 ^a
Short Description	cannot run the HtmlViewer or MarkupViewer in a stand-alone GUI

a. Available in https://bugs.eclipse.org/bugs/show_bug.cgi?id=245759.

The project task history is large since the projects last months, but for each performed task is possible to have the artifacts that were edited and by whom. Our method uses Clustering technique to organize the software project history database in clusters using edit interaction information. After, our method uses Natural Language Processing technique to identify the cluster that best fits the new task using textual information [20]. In this section we briefly introduce these two techniques.

Clustering is a computational technique for organizing data objects (elements) into groups (clusters) in order to provide an organization to support the human being in understanding information. Most similar elements tend to stay in the same group, while less similar or non-similar elements tend to stay in different groups [21], [22]. Similar groups do not have an identification according to the content of the groups, so this technique is also known as unsupervised learning.

The Natural Language Processing (NLP) is formed by a set of computational techniques motivated by theory for the automatic analysis and representation of human language [20]. The techniques and models designed for one language are not easily generalized to other languages in most cases [23]. According to Cambria and White [20], NLP research began with the word analysis paradigm, evolved to the analysis of concepts, and it is expected that models can understand narratives in the future.

III. METHOD TO RECOMMEND ARTIFACTS TO NEW TASKS

The Tacin method recommends a new task context defined as: 1- artifacts that will probably be edited by developers to perform a new task; 2- performed tasks similar to the new task; 3- experts in the new task. This paper presents the part of Tacin to recommends artifacts to new tasks in software projects.

In software projects, developers create or change artifacts to perform tasks. So, Tacin defines that the similarity weight between two tasks is equal to the number of artifacts edited in common between them. First, Tacin organizes task history into groups of similar tasks by edit interaction. Figure 1 shows the task representation of the Mylyn Docs project. Tasks are represented by green triangles; artifacts by blue squares;

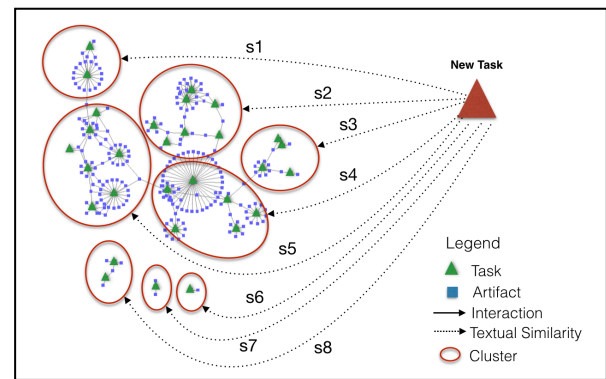


Figure 1. Illustration of the method to recommend artifacts to compose the context of a new task in software development project.

interactions by black lines linking tasks to their artifacts. A black line can represent one or more edit interactions on artifact.

The automatic determination of similar groups from a dataset is extensively studied in the literature [21]. Tacin uses LNS_SMC (Large Neighborhood Search - Software Module Clustering) heuristic based on the large neighborhood search metaheuristic. The LNS_SMC algorithm was chosen because presented a good efficiency using the Modularization Quality (MQ) measure applied to the clustering problem of software modules [20].

Figure 1 illustrates the association of a new task with the project task history organized in clusters. Each cluster can be seen as a single task where its *short description* is defined as the concatenation of the short descriptions of all tasks belonging to the cluster. Thus, the textual similarity between the new task and the clusters can be calculated.

Figure 1 also illustrates the recommendation of artifacts with Tacin. The first three groups (clusters) that present the greatest textual similarity to the new task are selected for the recommendation. Tacin recommends artifacts that belong to the group(s) considering three options: 1 - only the group most similar textually to the new task; 2 - the two groups most similar textually to the new task; 3 - and the three groups most similar textually to the new task. Tacin chooses a option according with the effectiveness of past recommendations. For this, the harmonic average between Reduction and Recall (Rc-measure) for three options is calculated for each new task, equations defined in (1-3). The option that has the highest Rc-measure receives 1 point. Then Tacin can recommend artifacts to developers using the highest scoring option. If there is a tie, Tacin recommends the option of a smaller number of selected groups.

$$\text{Reduction} = 1 - \frac{|\text{Recommended Artifacts}|}{|\text{Project Artifacts}|} \quad (1)$$

$$\text{Recall} = \frac{|\text{Recommended Artifacts} \cap \text{Relevant Artifacts}|}{|\text{Relevant Artifacts}|} \quad (2)$$

$$\text{Rc-measure} = 2 \times \frac{(\text{Recall} \times \text{Reduction})}{(\text{Recall} + \text{Reduction})} \quad (3)$$

IV. ASSESSMENT

The planning of the study includes objective, case, research questions and method according to the guide to conduct and report case study in Software Engineering [24]. The objective is described in the format indicated by GQM [25]: *Analyze the Tacin method to recommend artifacts for the purpose of evaluating their effectiveness from the perspective of the developer in the context of a software development project.* According to this objective following the RQ1 research question: Is the Tacin method effective in recommending artifacts at the beginning of a new software development task?

The effectiveness of Tacin depends on the effectiveness of task clustering by edit interaction and the correlation between textual similarity and similarity by edit interaction between tasks. Then two research sub-questions were defined: RQ1.A: Is task clustering by edit interaction effective for recommending artifacts? RQ1.B: Is there evidence on the existence of correlation between textual similarity and similarity by edit interaction among tasks?

Tacin reduces the number of artifacts available at the beginning of a new task, trying not to omit the artifacts that will be edited by developers. Equation 1 present Reduction measure. Equation 2 show Recall measure. So, Equation 3 combines these two measures in a harmonic way. Therefore, the Rc-measure metric was chosen to measure the efficacy of the Method.

The Mylyn Docs project was the case selected to evaluate the artifacts recommendation because it has the textual information of the tasks and developer interactions on the artifacts. The data collected from Mylyn Docs project was generated by the Mylyn, Git and Bugzilla tools. The Mylyn plugin logs developer's interactions about artifacts as interaction event with kind='edit'. The parameters for the query were Classification = Mylyn and Product = "Mylyn Docs" and Component = EPUB or Framework or HtmlText or Wikitext and Status = RESOLVE and Resolution = FIXED and Match ALL of the following separately → Attachment Description → contains the string → mylyn/context/zip. The query was executed in the site <https://bugs.eclipse.org/bugs/query.cgi> at April 05, 2017 and returned 334 tasks with 49906 edit interactions performed by 6 developers over 1538 artifacts. We have identified many performed tasks in the Mylyn Docs project that do not have Mylyn logging. So, committing actions were collected to infer editing actions, usually, a committing action submits edited files to code repository. Commits extraction resulted in 918 commits performed by 33 developers over 5407 artifacts.

The study generated artifact recommendations to new task in 20 trials, simulating a new task on each first day of the month from 09/01/2008 to 04/01/2010 to evaluate a long time period. The artifacts that were associated with tasks up to the date of the trial and were also associated with the new task after their completion form the set of relevant artifacts of the new task. Tacin tries find these relevant artifacts using only edit interactions finished before the start of each new task. Each trial generates groups of similar tasks by edit interaction. For each trial, 30 trials using the LNS_SMC were performed and the cluster with the highest MQ was chosen to be used to make recommendation.

Tacin calculates all textual similarities between the new task and the calculated clusters. The *short description* field of the new task was used with concatenation of all the *short description* fields of the tasks in each cluster. RapidMiner Studio was used to calculate textual similarity. The text processing used the functions Replace Tokens, Transform Cases, Tokenize, Filter Stopwords, Filter Tokens (by Length) and Stem (Porter). Then the texts were represented in vectors using the occurrence of terms. Finally, the similarities between the vectors (texts) were calculated using the cosine similarity function.

The evaluation contemplates three options of recommendation according to textual similarity among new task and tasks clustered by edit interaction. The first option evaluates the recommendation of the artifacts to the group that presents greater similarity. The second recommends the task artifacts of the two groups that present the greatest similarities. Likewise, the third one recommends the task artifacts of the first 3 groups. These 3 options are rated according to Recall, Reduction and Rc-measure for each trial.

A. Results

The average Recall of the first option was 37%, with Reduction equal to 97% and Rc-measure of 41%. The second option presented Recall equal to 45%, Reduction of 93% and 48% of Rc-measure. The third one obtained average Recall of 52%, Reduction of 90% and 57% of Rc-measure. Accordingly, we observed that the third option is the best according to the average values of Rc-measure.

The result of this study shows that clustering of similar tasks by interaction (RQ1.A) built at least one cluster that had only 6% (94% of Reduction) of the artifacts available in the project, but that had 77% of relevant artifacts (Recall) for a new task. The conclusion is that the answer is yes to RQ1.A when combined LNS_SMC with MQ. In 8 rounds, the first cluster more similar textually with the new task also presented the highest Recall. However, in 10 trials, the 3 clusters most similar textually to the new task did not present maximum Recall, among these, in 4 trials (20%) there are no correlation between similarity by interaction and textual. The conclusion is that the answer is yes also for RQ1.B, the study showed that there is evidence of correlation between textual similarity and similarity by interaction in software development tasks.

V. RELATED WORK

Hipikat uses a large number of documents available in the project such as source code, documentation, communications between developers (e-mail, discussion forums), error reporting and test plans. The Hipikat evaluation presented an average Recall of 65% [1]. Antunes et al. [8] proposed a recommendation system to recommend a list of relevant artifacts. The System uses developer interactions in real time and artifact access time to list the most relevant artifacts. Then it uses the relations of the language structure (Java) and textual associations to order the recommended artifacts in a decreasing way of relevance. The evaluation showed an average Recall of 42,7%.

The proposal of Ye et al. [9] lists a classification of relevant artifacts (top 10) to correct an error in the software considering the information of the project history. The evaluation indicates

that the proposal can recommend relevant artifacts in 70% of the recommendations. CodeRants is a recommendation method for recommending artifacts. This Method is based on the repetition of the textual searches performed by the programmers and on the metaphor of the ant colony [10]. The evaluation was performed in a simulated environment, in this environment CodeRants obtained an average Recall of 71%. Almhana et al. [11] have shown that to recommend relevant artifacts to support the developer in the correction of an error can be mitigated as a multi-objective problem, maximizing the relevance of the recommended artifacts and minimizing the number of recommended artifacts. The evaluation showed strong evidence that the proposal may recommend lists with relevant artifacts: Recall @ 5 = 72%; Recall 10 = 81%; Recall 15 = 87%; Recall 20 = 94%.

VI. CONCLUSION

This paper presents part of Tacin method to recommend artifacts to compose the context of new tasks in software projects. Tacin makes use of Clustering and Natural Language Processing techniques. The clustering of task history in similar tasks by interaction presented a Recall of 77%. The next study will consider the degree of relevance between artifact and task. The textual association of the new task with the similar task groups to produce artifacts recommendation with 52% of Recall. This study uses only task *short description* field. The next study should consider other text information from task history, e.g., comments from developers while performing tasks. In addition, other techniques such as dw-cosine [26], an extension of the cosine of similarity, and also techniques for small texts with grammatical errors need to be evaluated.

ACKNOWLEDGMENT

This work was partially supported by the Brazilian funding agencies CAPES and CNPq and Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

- [1] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth, "Hipikat: a project memory for software development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 446–465, Jun. 2005.
- [2] M. Kersten, "Focusing knowledge work with task context," University of British Columbia, 2007.
- [3] D. W. McDonald and M. S. Ackerman, "Expertise Recommender: A Flexible Recommendation System and Architecture," in *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, New York, NY, USA, 2000, pp. 231–240.
- [4] Z. Wang, H. Sun, Y. Fu, and L. Ye, "Recommending crowdsourced software developers in consideration of skill improvement," presented at the ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, 2017, pp. 717–722.
- [5] I. Roediger Henry L., "Relativity of Remembering: Why the Laws of Memory Vanished," *Annu. Rev. Psychol.*, vol. 59, no. 1, pp. 225–254, Dec. 2007.
- [6] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, Eds., *Recommendation Systems in Software Engineering*. Berlin Heidelberg: Springer-Verlag, 2014.
- [7] M. Andric, W. Hall, and L. Carr, "Assisting artifact retrieval in software engineering projects," presented at the Proceedings of the 2004 ACM Symposium on Document Engineering, 2004, pp. 48–50.
- [8] B. Antunes, J. Cordeiro, and P. Gomes, "An Approach to Context-based Recommendation in Software Development," in *Proceedings of the Sixth ACM Conference on Recommender Systems*, New York, NY, USA, 2012, pp. 171–178.
- [9] X. Ye, R. Bunesco, and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge," presented at the Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2014, vol. 16-21-November-2014, pp. 689–699.
- [10] I. Caicedo-Castro and H. Duarte-Amaya, "CodeRants: A recommendation method based on collaborative searching and ant colonies, applied to reusing of open source code," 2015.
- [11] R. Almhana, W. Mkaouer, M. Kessentini, and A. Ouni, "Recommending relevant classes for bug reports using multi-objective search," presented at the ASE 2016 - Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, 2016, pp. 286–295.
- [12] A. Moraes, E. Silva, C. da Trindade, Y. Barbosa, and S. Meira, "Recommending Experts Using Communication History," in *Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering*, New York, NY, USA, 2010, pp. 41–45.
- [13] T. Fritz, G. C. Murphy, E. Murphy-Hill, J. Ou, and E. Hill, "Degree-of-knowledge: Modeling a developer's knowledge of code," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 2, 2014.
- [14] X. Xie, W. Zhang, Y. Yang, and Q. Wang, "DRETOM: Developer Recommendation Based on Topic Models for Bug Resolution," in *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, New York, NY, USA, 2012, pp. 19–28.
- [15] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," presented at the Proceedings - Working Conference on Reverse Engineering, WCRE, 2013, pp. 72–81.
- [16] J. Zhu, B. Shen, and F. Hu, "A learning to rank framework for developer recommendation in software crowdsourcing," presented at the Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2016, vol. 2016-May, pp. 285–292.
- [17] Y. Malheiros, A. Moraes, C. Trindade, and S. Meira, "A Source Code Recommender System to Support Newcomers," in *2012 IEEE 36th Annual Computer Software and Applications Conference*, 2012, pp. 19–24.
- [18] B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala, "DebugAdvisor: A recommender system for debugging," presented at the ESEC-FSE'09 - Proceedings of the Joint 12th European Software Engineering Conference and 17th ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2009, pp. 373–382.
- [19] M. C. Monçores, A. C. F. Alvim, and M. O. Barros, "Large Neighborhood Search applied to the Software Module Clustering problem," *Computers & Operations Research*, vol. 91, pp. 92–111, Mar. 2018.
- [20] E. Cambria and B. White, "Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]," *IEEE Computational Intelligence Magazine*, vol. 9, no. 2, pp. 48–57, May 2014.
- [21] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data Clustering: A Review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999.
- [22] J. Han, M. Kamber, and J. Pei, "10 - Cluster Analysis: Basic Concepts and Methods," in *Data Mining (Third Edition)*, Third Edition., J. Han, M. Kamber, and J. Pei, Eds. Boston: Morgan Kaufmann, 2012, pp. 443–495.
- [23] R. Levy and C. Manning, "Is It Harder to Parse Chinese, or the Chinese Treebank?," in *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, Stroudsburg, PA, USA, 2003, pp. 439–446.
- [24] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, p. 131, Dec. 2008.
- [25] V. R. Basili, "Software Modeling and Measurement: The Goal/Question/Metric Paradigm," University of Maryland at College Park, College Park, MD, USA, 1992.
- [26] B. Li and L. Han, "Distance Weighted Cosine Similarity Measure for Text Classification," in *Intelligent Data Engineering and Automated Learning - IDEAL 2013*, 2013, pp. 611–618.

SOTagger - Towards Classifying Stack Overflow Posts through Contextual Tagging

Akhila Sri Manasa Venigalla
Indian Institute of Technology
Tirupati, India
cs18m017@iittp.ac.in

Chaitanya S. Lakkundi
Indian Institute of Technology
Tirupati, India
cs18s502@iittp.ac.in

Sridhar Chimalakonda
Indian Institute of Technology
Tirupati, India
ch@iittp.ac.in

Abstract—There is an ever increasing growth in the use of Q&A websites such as Stack Overflow (SO), so are the number of posts on them. These websites serve as knowledge sharing platforms where Subject Matter Experts (SMEs) and developers answer questions posted by other users. It is effort intensive for developers to navigate to right posts because of the large volume of posts on the platform, despite the presence of existing tags, that are based on technologies. Tagging these posts based on their context and purpose might help developers and SMEs in easily identifying questions they wish to answer and also in identifying contextually similar posts. To support this idea, we propose *SOTagger* as a prototype plug-in for Stack Overflow to tag questions contextually. We have considered SO data provided on *SOTorrent* and automated the identification of 6 categories of questions using Latent Dirichlet Allocation. We have also manually verified relevance of these categories. Using these categories and dataset, we have built a classification model to classify a post into one of these six categories using Support Vector Machine. We have evaluated *SOTagger* by conducting a user survey with 32 developers. The preliminary results are promising with about 80% developers recommending the plugin to others.

Index Terms—Stack Overflow, Contextual Tagging, LDA

I. INTRODUCTION

Stack Overflow (SO) is one of the most frequently used websites with about 11M visits every day. With a user base of 10M users, about 7.3K questions are posted per day. It comprises of about 18 million questions, of which 71% are answered¹. These questions correspond to various technical categories, tools, libraries and are tagged into atmost 5 of 54K tags² present on the website. This tagging is done based on their technical relevance with the posted content and is used to organize posts and thus help users to browse for questions and answers concerning to particular topics such as *javascript*, *jquery*, *python* and so on [1]. However, these tags don't classify questions based on the context in which they are asked. The context would capture situations pertaining to conceptual understanding, issue resolving and so on.

Recent studies have aimed at classifying questions on SO based on their context and arrived at almost similar

taxonomies of categories. They have used various techniques such as K-NN clustering [2], automatic categorization by topic modeling using LDA and MALLET [3] and manual categorizations [1], [4]. Some of these studies have aimed to contextually categorize technology-specific questions such as questions related to Android application development [2] and mobile operating systems like *Android*, *Apple* and *Microsoft Windows*. However, existing tools do not categorize posts on SO platform based on context. To this end, the contributions of this paper are as follows:

- *SOTagger*³ - a prototype plug-in that classifies posts on SO into six categories: *Conceptual*, *Discrepancy*, *Implementation*, *Error*, *Learning* and *MWE (Minimum Working Example)*.
- Application of NLP techniques - Latent Dirichlet Allocation(LDA) and Machine learning (ML) classifier - Support Vector Classifier (SVC) to classify SO posts.
- Evaluation of *SOTagger* with 32 professional developers and manual cross-verification of 100 posts.

II. RELATED WORK

In the recent years, several studies have been done to analyze posts on SO, which include analyzing developers' area of interest based on questions asked [5], analyzing and suggesting tags of the questions [2] [1] [6] [7], identifying difficulties faced by developers [8], identifying trending technological topics [9], and so on. Researchers have classified posts on SO based on the context by manually interviewing software developers. In a survey conducted by Latoza et al., 179 professional software developers were asked to identify hard-to-answer questions pertaining to code that they solicit wherein 371 questions were reported. They have manually categorized them into 21 categories with 94 distinct questions, of which the 5 most frequently reported categories were - *Rationale*, *Intent* and *Implementation*, *Debugging*, *Refactoring* and *History of code* [10].

Studies have been conducted to investigate various question categories based on the context in which they

DOI reference number: 10.18293/SEKE2019-067

¹<https://stackexchange.com/sites?view=list#traffic>

²<http://bit.ly/SONumTags>

³<https://github.com/chaitanya-lakkundi/SOTagger>

were asked. Rosen et al. manually categorized 380 posts on SO into 3 question categories based on the three interrogative words- *How, What and Why*, corresponding to three mobile operating system categories - *Android, Apple and Microsoft Windows* [4]. Treude et al. have manually classified 385 questions on SO into 10 categories - *How to, Decision Help, Discrepancy, Environment, Error, Conceptual, Review, Non-Functional, Novice, Noise* [1]. Although methods involving manual effort are necessary to capture ground truth, we see a need to find better ways to scale this approach such that automation is possible.

Elucidating further studies, Beyer et al. have proposed 7 question categories - *API Change, API Usage, Conceptual, Discrepancy, Learning, Errors, Review* by manually classifying 500 SO Android posts and performed automatic classification using supervised machine learning algorithms with a precision of 88% [2]. Allamanis et al. found 5 major question categories using LDA and unsupervised machine learning algorithm [3].

Insofar as the development in methods of classification is concerned, the research community has progressed from significant manual studies to automating them using machine learning algorithms and NLP techniques. Contemporary tools such as EnTAGREC++ [6], TagCombine [7] have been developed to provide tag suggestions to users when they post questions on SO. These tools suggest tags based on technologies involved in the post content. The prototype plug-in we propose, *SOTagger*, tags posts on SO based on their purpose or intent rather than considering the technologies involved. Based on the existing work on classifying posts [2] [1] [4] [3], we propose a taxonomy to tag posts contextually.

III. PROPOSED TAXONOMY

Posts can be classified using several NLP techniques such as LDA, LSA, TF-IDF. However, inline with the existing work, we followed LDA technique.

We present six question categories that we have derived from existing studies and results obtained from LDA topic modeling. As a result of LDA topic modeling configured for 6 topics, we obtained 6 topics characterized by keywords for each topic, along with the weightage of keywords in every topic. Omitting the technical terms and considering interrogatives, it has been observed that *Topic 0* comprises of *discrepancy*, *Topic 1* contains *error*, *Topic 2* contains *how-to or implementation*, *Topic 3* contains *learning*, *Topic 4* contains *conceptual* and *Topic 5* contains *MWE* keywords respectively, as shown in Table I. These results obtained by applying LDA on SO posts indicate the presence of contextual categories in SO data. Comparing these results with the existing taxonomy discussed by Beyer et al. in [2] and other taxonomies presented in [1] [4] [3], we reorganize few categories in the existing literature and arrive at labelling five of these six topics as *conceptual, discrepancy, implementation, error and learning* respectively. We

TABLE I
TAXONOMY OF QUESTION CATEGORIES

S.No.	Topics	Keywords
1	Conceptual	What is use/difference, Is there a way, Is it possible[2]
2	Discrepancy	doesn't work, tried to, have/facing problem, before upgrade previous version [2]
3	Implementation	How to implement [4] [3] [1]
4	Error	Exception, error [2]
5	Learning	suggest, tutorial, where can I find [2]
6	MWE	for this code, code tags

observed that many of the posts on SO contained code snippets, which could indicate that users post questions containing code to reproduce the bug they are facing. Such code snippets serve as *Minimum Working Examples (MWE)*⁴, which is proposed as another category *MWE*. We observe this naming to be inline with work proposed by Allamanis et al. [3]. Each post can be classified into one or more of these six categories.

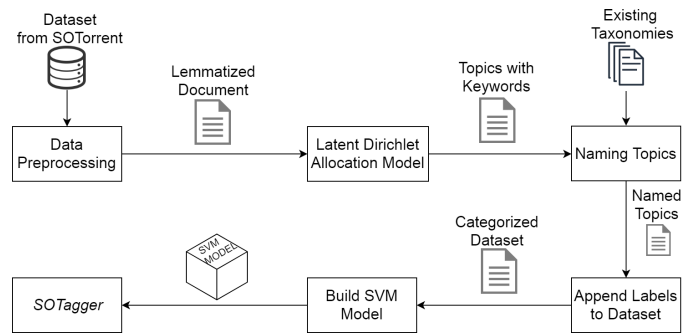


Fig. 1. Overview of Approach for *SOTagger*

IV. DESIGN METHODOLOGY

We followed a six step approach in designing a contextual classification model as shown in Fig 1.

Step 1 - Extract DataSet. To perform categorization of SO posts, we downloaded *Posts.xml* file available on *SOTorrent*⁵. We considered a subset of this file that constituted 100K Stack Overflow posts under *Body* column and filtered out questions based on *PostTypeId* column that resulted in a dataset of 20K posts.

Step 2 - Data Preprocessing. Data present in *Body* column whose *PostTypeId* = 1 was considered for pre-processing. We considered English stop words provided by NLTK library and omitted interrogative words from the list of stop words keeping in view, the taxonomy proposed. We processed the data for stop word, punctuation removal and lemmatization using *spaCy*.

⁴<https://stackoverflow.com/help/mcve>

⁵<https://zenodo.org/record/2273117>

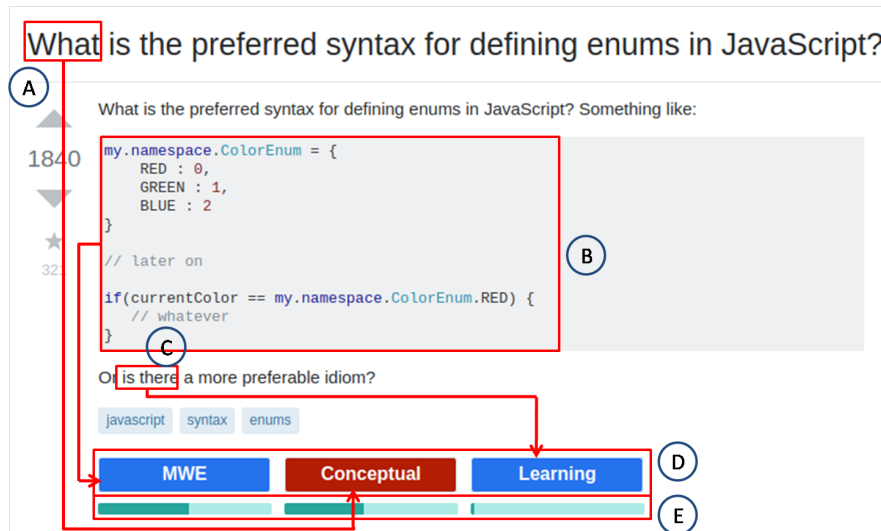


Fig. 2. A Snapshot of *SOTagger*

Step 3 - Latent Dirichlet Allocation Model. We applied LDA to perform topic modeling. We primarily created a dictionary of lemmatized words and then created a corpus of these words with their frequency of occurrence. Considering this corpus, we generated an LDA model that categorizes given data into 6 topics.

Step 4 - Naming Topics. Based on existing taxonomies in the literature [2] [1] [4] [3], we identified contextually useful keywords in each of the 6 topics, and used them to identify and name topics.

Step 5 - Append Labels to Dataset. The LDA model provided us with a topic-document correlation matrix, where document refers to content of one post. This matrix contained probabilities of every identified topic for each document. We then classified posts in the dataset into topics based on the dominant topic from correlation matrix which had the highest probability.

Step 6 - Prepare a Machine Learning model - Build SVM Model. We applied various machine learning classification algorithms such as *Linear SVC*, *Logistic Regression*, *Multinomial Naive Bayes*, *Random Forest Classifier* to arrive at the best classification model on available dataset with 75% train and 25% test data. We observed that SVC was able to classify the given data set with higher accuracy (78.5%) than other models. Based on this, we designed SVC model and pipelined to *CalibratedClassifierCV* to get prediction probabilities.

V. DEVELOPMENT OF *SOTagger*

This plug-in has been developed as an extension to *Google Chrome* to support classification of posts on SO. It tags posts on SO based on their context. *SOTagger* reads SO posts on the page and extracts questions from these posts which are fed into previously developed ML classification models using SVM classification. This model outputs the categories of specific posts along

with associated probabilities which are presented as tags below the posts on SO platform.

A snapshot of *SOTagger* is shown in Fig 2 for a sample post on SO. Tags corresponding to context of the question are displayed below the post as shown in [D] of Fig 2 and are arranged in decreasing order of probability. The probability with which a post is tagged into each of the displayed categories is represented by a bar as depicted in [E] of Fig 2. According to *SOTagger*, this post is classified as *MWE* category with highest probability. As pointed in [B] of Fig 2, presence of code segment justifies classification of the post into *MWE* category. Presence of *What* keyword as highlighted in [A] of Fig 2, contributes to *Conceptual* tag, with a lesser probability than *MWE* tag. *is there* phrase represented by [C] of Fig 2 contributes to *Learning* category, with least probability.

However, the keywords or phrases demonstrated in Fig 2, are for the purpose of analyzing the correctness of *SOTagger*, but are not the only basis for classification. Actual classification was based on NLP and ML techniques that have been used in development of *SOTagger*.

VI. EVALUATION AND RESULTS

We evaluated *SOTagger* by conducting a user survey with 32 professional developers with a development experience ranging from 2 years to 19 years.

The participants were asked to use *SOTagger*, navigate to SO website and analyze the contextual tags added by *SOTagger*. A user survey was conducted with the help of five point Likert scale, containing a questionnaire as provided in Table II.

Apart from user survey, we manually evaluated⁶ contextual tags of about 100 random posts on SO tagged by *SOTagger* and obtained an accuracy of 77%. The

⁶<https://git.io/fjC83>

results of our survey indicate, *SOTagger* had a good user-friendly interface (82% in Q1). In Q2, about 85% of participants have agreed that *SOTagger* has appropriately tagged the posts. The ratings in Q3 and Q4 indicate that *SOTagger* has helped about 80% of participants in faster browsing of posts on SO and that the experiment has been considerably interesting (81% in Q4). In Q5, most of the participants have agreed that they would recommend *SOTagger* to their peers (83%).

TABLE II
QUESTIONS IN SURVEY USING A 5-POINT LIKERT SCALE.

Q1: How easy was it to use <i>SOTagger</i> interface?
Q2: <i>SOTagger</i> has tagged SO posts correctly based on their context.
Q3: <i>SOTagger</i> has helped me in quick browsing of posts based on context.
Q4: <i>SOTagger</i> has kept the whole experiment interesting and informative.
Q5: I will recommend <i>SOTagger</i> to my peers.

VII. THREATS TO VALIDITY

We have manually examined top 20 posts based on probability values in each of the 6 topics generated by LDA technique to assign topic name. This could be inaccurate considering limited number of posts examined.

To understand the accuracy of classification, we randomly browsed 100 posts on SO. We realize that examination of 100 posts in total is not enough to get an overall idea about the accuracy of classification. During the creation of LDA model, we tweaked a few parameters such as chunk size and number of passes which resulted in different statistical distribution of topics. Some of the distributions were imbalanced and biased towards one particular topic. We selected those parameters which resulted in a nearly Gaussian distribution. We assume that LDA model which classifies data in Gaussian distribution performs better than other models. However, initial results show that accuracy of trained LDA model is around 70%, but with scope for experimenting with other distributions. The machine learning model has been trained on a dataset of 20K questions, however we should consider a larger number of posts from SO to improve our approach.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we presented *SOTagger*, a prototype plug-in to SO that tags questions on SO based on the purpose for which they are asked. We performed LDA topic modeling on data set available on *SOTorrent* to identify categories. We labelled the resultant LDA topics by harmonizing the existing taxonomies. We presented 6

question categories, independent of technical aspects involved in the questions. We then labelled question posts in the dataset into one or more of the 6 categories. We applied SVC on the labelled dataset to obtain machine learning classification model which was integrated into the plug-in to support tagging of posts on SO.

As a part of future work, we plan to extend *SOTagger* to display contextual tags of posts on SO landing page by training machine learning model only over titles of questions. We plan to work in the direction to improve levels of taxonomy from single level presented in the paper to multiple levels and display the same as a part of detailed contextual tagging. We could conduct an experiment to check whether we get better results by considering the opening and closing statements of SO posts.

Questions tagged with MWE could be of greater use for future research. Researchers interested to understand and analyze code provided by users when posing questions can easily find questions with this tag. We envision that future work based on this paper may include clustering posts classified as MWE to automatically find bugs, combine co-occurring tags to formulate new tags and so on. Also, several empirical studies on SO posts such as understanding code quality, misuse of code snippets and automatic bug reporting could be conducted.

REFERENCES

- [1] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web?: Nier track," in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 804–807.
- [2] S. Beyer, C. Macho, M. Pinzger, and M. Di Penta, "Automatically classifying posts into question categories on stack overflow," in *Proceedings of the 26th Conference on Program Comprehension*. ACM, 2018, pp. 211–221.
- [3] M. Allamanis and C. Sutton, "Why, when, and what: analyzing stack overflow questions by topic, type, and code," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 53–56.
- [4] C. Rosen and E. Shihab, "What are mobile developers asking about? a large scale study using stack overflow," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016.
- [5] R. K.-W. Lee and D. Lo, "Github and stack overflow: Analyzing developer interests across multiple social collaborative platforms," in *International Conference on Social Informatics*. Springer, 2017, pp. 245–256.
- [6] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "Entagrec ++: An enhanced tag recommendation system for software information sites," *Empirical Software Engineering*, vol. 23, no. 2, pp. 800–832, 2018. [Online]. Available: <https://doi.org/10.1007/s10664-017-9533-1>
- [7] X.-Y. Wang, X. Xia, and D. Lo, "Tagcombine: Recommending tags to contents in software information sites," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 1017–1035, 2015.
- [8] A. Joorabchi, M. English, and A. E. Mahdi, "Text mining stack-overflow: An insight into challenges and subject-related difficulties faced by computer science learners," *Journal of Enterprise Information Management*, vol. 29, no. 2, pp. 255–275, 2016.
- [9] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.
- [10] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code," in *Evaluation and Usability of Programming Languages and Tools*. ACM, 2010, p. 8.

An Annotated Repository for MATLAB Code

António Relvas
NOVA LINCS
DI-NOVA/FCT
Portugal

Nuno C. Marques
NOVA LINCS
DI-NOVA/FCT
Portugal
Email: nmm@fct.unl.pt

Miguel P. Monteiro
NOVA LINCS
DI-NOVA/FCT
Portugal
Email: mtpm@fct.unl.pt

Glauco Carneiro
Universidade Salvador
UNIFACS
Brazil
Email: glauco.carneiro@unifacs.br

Abstract—Currently, there is the need for systems to manage repositories of MATLAB code bases capable of supporting global queries and feed their results to analyses components. Such features are not directly supported in current platforms. This paper presents a repository management system that supports queries over semi-automatically annotated code files and are able to associate them to higher level concepts. To meet this need, this paper proposes an approach that equips the repository with support for sophisticated queries over its stored code base and allows patterns to emerge from such queries, namely for visualisation and further analysis. This is achieved through the synergistic combination of a token-based metrics extraction component and a relational model fed by an ubiquitous data mining process. The code base is represented by means of relational knowledge, enabling intelligent queries that can be extended with new code metrics. Presently, query results are being used for the detection of concerns, including those whose code is scattered over multiple modular units. This paper outlines the proposed system's architecture and presents a proof-of-concept implementation developed for MATLAB programs. It is evaluated by means of a set of illustrative queries over a seed repository of MATLAB systems.

Index Terms—MATLAB, Concern, Self Organizing Map, UbiSOM, Advanced Data Exploration, Software Repository Management System

I. INTRODUCTION

The MATLAB language is known to lack support for fully fledged modules capable of enclosing most concerns typically present in MATLAB systems [1] [2]. Its basic units of modularity are *m-files* (MATLAB code files) and *toolboxes*, comprising folders of *m-files* and optionally sub-folders. The latter often correspond to standalone programs or libraries made available for the user community. Lack of modularity makes it hard to obtain well-organised code that is easy to read and reuse [1] [3]–[5]. These shortcomings motivated ongoing research to study the symptoms induced by the lack of modularity [3] and use of that knowledge for the detection of unmodularised concerns in existing systems [6] [5].

The above research on concern detection techniques is centred on the idea of decomposing *m-files* into its low-level elements [3] and derive a number of metrics based on that information [6] [5]. A number of analysis components can subsequently be plugged into the system to derive higher-level information. Further details are given in section II.

Developing and maturing the concern detection techniques entailed the assembling of repositories of many MATLAB

systems in order to exercise and test functionalities. A number of metrics were derived, which were extracted from the code base by means of a tool that includes a lexical analyser for MATLAB [3] [6] [5]. Activities on the code repository included the performing of many kinds of search, e.g., to find new patterns, perform studies, check results. Initially, it was carried out by the metrics extraction tool. As search patterns tended to become increasingly elaborate and structured, the motivation for mounting the entire repository in a more intelligent system arose. This paper presents the outcome of that effort.

This paper presents a repository management system that exposes low-level data and allows the plugging of new analysis and visualisation components, which can be added over time. The repository management system could compute many of the queries previously supported by the lexical analyser tool and much else besides. It supports queries over automatically added annotations received from a data mining process and related annotations made by humans. Among other things, the system is able to expose associations between *m-files* based on higher level concepts.

Past work illustrated one potential use of the proposed repository through the implementation of a high level knowledge from a *Self Organizing Map* (SOM) data mining model that derives patterns from the code to enable the semi-automatic detection of (possibly unmodularised) concerns [6] [5]. By *semi-automatic*, we mean a process that starts with a machine learning classification phase, whose output can be corrected by a human or enriched by new annotations. We implemented a relational database and an associated web interface through which query results produced by the UbiSOM algorithm [7] are made available for further processing.

The rest of this paper is organised as follows. Section II describes the token-based technique used to decompose *m-files*. Next, section III describes the system's architecture and the relational model that enables a wide variety of queries over the code. Section IV illustrates how an external concern mining model can be used to automatically provide concern relevant tagging for all *m-files* in the repository. Section V presents a number of queries that illustrate the repository's capabilities. Section VI provides a short discussion of this work and section VII concludes the paper.

II. USE CASE: CONCERN DETECTION

This section relates to past research on the detection of concerns in MATLAB code bases [3] [5] and describes an use case for the annotated repository.

A *concern* is any abstraction, concept or cohesive set of functionalities that would ideally be enclosed in its own module, for the sake of comprehensibility and ease of maintenance and evolution [8]. Ideally, each individual concern would map to a different unit of modularity (e.g., an m-file or a function in MATLAB), with each unit having a single, *primary* concern. However, several factors contribute to this not being so in practice. The limited and incipient nature of MATLAB's modules and limited programming experience from many users, among other issues, contribute to many concerns remaining unmodularised. As a consequence, potentially useful and reusable pieces of code are left *scattered* throughout a system's m-files and functions, and *tangled* with conceptually unrelated code. Scattering and tangling are the two dual symptoms usually observed in the code when modularity support is deficient [9] [8]. Tangling is particularly harmful to the comprehensibility of all concerns found in the modular unit, including the primary concern [1] [4] [3].

The concern detection technique explored in this paper bases the representation of an entire code base of a repository, comprising all systems stored in it, on the decomposition of each and every code file into *tokens*, i.e., the lexical elements extracted by means of the lexical analyser. The subset of tokens that are *words* (keywords and identifiers) plays an important role in the concern detection approach. It is based on the idea that specific groups of word tokens can be associated to specific concerns, with individual tokens being associated to one concern at most. Patterns of occurrence of such tokens can be used to identify the presence of the associated concern in the code unit.

Presently, this approach is focused on function names, particularly names of functions from standard MATLAB libraries, because they are deemed more intention-revealing and are common to many MATLAB systems. Such names provide a measure of guarantee that the technique will operate uniformly in most systems. Programmer defined variable names are not currently considered because they are more variable across a repository of systems developed by many different teams.

The examples shown in section V serve as an illustration of the technique. They are focused on concern *verification of function arguments*, which relates to the processing that many MATLAB functions must carry out at the beginning to determine in which "mode" they were called (e.g., by finding out how many arguments it received). It is associated to a group of tokens that include `nargin`, `varargin` and `varargout`.

III. THE RELATIONAL MODEL

This section describes a relational model for a MATLAB seed code repository. It is important to note that although the system presented in this paper was developed as a proof of concept focused on MATLAB systems, its design – described

here – was created in view of covering a broader range of systems and programming languages.

The annotated repository was developed with the purpose of accommodating in one place all the data needed to perform exploratory analyses on MATLAB code, by means of advanced analysis components. The diversity and quantity of data being generated called for a powerful solution for data storage and query support. For this reason, a relational data model was adopted (e.g., [10]). It represents all data as *tuples*, grouped into relations. Note that *all* tokens are stored into the model: not just word tokens but also symbolic tokens, literals, etc. Such a model can easily be implemented in a relational database management system. A MySQL solution proved to be sufficient for the Web component of the system, while data analysis is based on snapshots of the systems in the repository, taken at the time of analysis of relevant information stored in a SQLite database.

The relational model represents the toolboxes, m-files and their complete contents (including comments, though presently they are not used). New systems, toolboxes and m-files can be added and similarly represented, including new versions of existing elements. Two main entities represent the organisation of all the systems into toolboxes and m-files and taking into account the possibility of multiple versions of these elements. Each m-file is in turn decomposed into code lines containing MATLAB code and thus stored. To reconstruct the original file (e.g., for visualisation or for some other subsequent processing), the model provides a separate entity to represent lines with comments only.

Tokens are the main subject of the analysis in this paper. Figure 1 depicts the main entities and relations for modeling the relation of MATLAB tokens and annotations within the repository according to the notation used by Silberschatz et. al. [11]. *Blocks* are intermediate entities grouping one or several lines and that are part of an m-file. Relations between toolboxes, m-files, code blocks, lines and tokens are simple one-to-many relations. For instance, a given line is always part of a block, which is always part of some m-file. Though an m-file is not necessarily organized into functions, we are mainly interested in that set of m-files since our focus is on MATLAB modularity. Under that view, blocks will always belong to some function, which in turn always belongs to some m-file. Decomposition of the entire code base along these lines requires just some basic parsing functionality added to the lexical analyser tool. Each block ends either with the `end` keyword or when another clearly defined block (e.g., a new function or control structure) begins. During design, we chose to also model each line of code in the repository as a tuple of the entity `Lines_mfiles` identified by its unique `line_id` identification code. The line number within the m-file and the block unique id are also unique identifiers for each line in the repository.

A token *instance* refers to individual occurrences of a given token. For instance, the various occurrences in the code of the `while` keyword can be said to be instances of the `while` token. The present token-based approach requires the

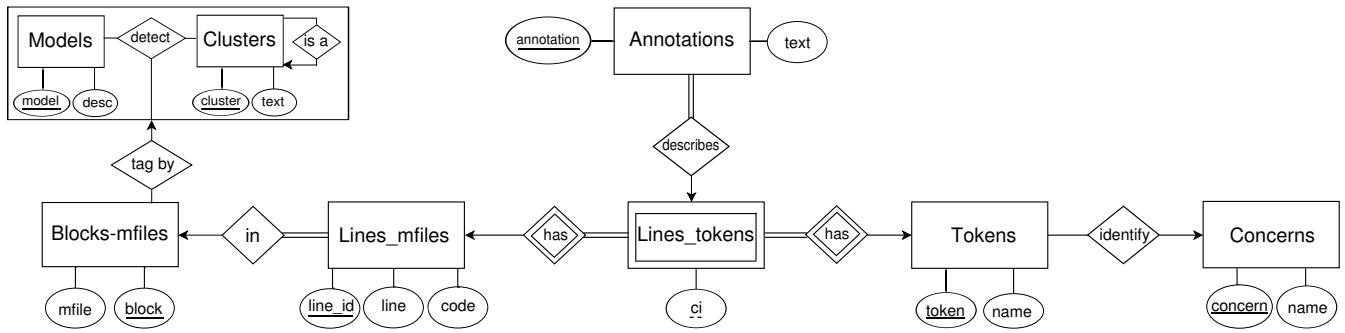


Fig. 1. Part of the system ER diagram for token related entities.

identification of each token instance, which is done by the lexical analyser tool. Each line comprises a *sequence* of token instances (order and position are important). Each non-empty line comprises at least one token instance or is a pure comment line in Fig. 1. Since a line can contain more than one instance of a given token, a weak entity (*Lines_tokens* in Fig. 1) is used to represent it. That weak entity is identified by the *line_id* of the *Lines_mfiles* entity and token position within the line (attribute *ic*). So, it is possible to know the line containing the token instance (attribute *line* from entity *Lines_mfiles*). This way, an indirect relation is made where each token instance is associated to its code or name. A token can also be associated to a given concern and a concern can be associated to several tokens — represented by the *identify* relation in Fig. 1. The design takes into account that future analyses may approach the code repository based on many different selection criteria. Annotations are also easily added using the *Lines_tokens* weak entity. This way a new entity *Annotations* is used so that users can add annotations when relevant *Lines_tokens* are found in Fig. 1. A simple annotation revision process is already supported: only accepted annotations will be shown in the final user interface.

During tool tests there was often a need to look for different combinations of two or more tokens or token-concern combinations. For instance, it was useful to find and visualize in what context some combinations occur in different m-files. The query can be done directly with a regular expression on the *code* field of entity *Lines_mfiles*, but this search is slow and slow to write. Besides, it will always be ineffective because limited to a single code line. As an alternative, the web interface performs a quick generation of queries in SQL (over the relational model). To obtain results quickly, the search is done on a first token after which an inner join is added for each additional token, of the resulting relation with itself. This way, it is possible to identify a sequence of elements (tokens or concerns) that are associated in the same m-file (through entity *Lines_tokens*). If they are relevant, extra constraints can be added to the query. For example, it is possible to specify a limit on the number of rows (or tokens) between the occurrences of both search tokens (the *ic* field of entity

Lines_tokens is essential for a correct result).

The code base used as testing material for the present research originates from a repository of MATLAB programs and toolboxes originally assembled to test a compiler for MATLAB [12]. It comprises 35 193 m-files organised by toolboxes and covering various application domains, downloaded from *Sourceforge* and *GitHub* [12]. The repository was already used in previous work for concern mining [6] [5].

The output of a fully automatic concern mining component can be easily related with this system. The relational model aims to be generic and should support hierarchical unsupervised machine learning methods. The simultaneous reference to several concern mining models is supported by means of an aggregation between entities *Models* and its detected *Clusters*. Such aggregation can be used for tagging *Blocks-mfiles* entries with the cluster assigned by a model, resulting from a data mining process over the repository. Also some unsupervised learning methods discover models where clusters can be related with other clusters (e.g. in hierarchical clustering methods). This way, each block in the repository can be assigned to a cluster derived from the concern mining model and the resulting cluster can be a sub-cluster of another related cluster. Moreover, no restriction is made regarding sharing of clusters among models (this could be useful in situations where related models identify the same kind of clusters). Finally, *Block-mfiles* tagging can be continuously updated by a ubiquitous data mining process (such as illustrated in section IV) or can be directly assigned/revised by means of a human made model (which probably entails laborious manual m-file cluster identification and correction tasks). The concern mining model presented in previous work [5] is used in the illustrative results presented in section V.

IV. VALIDATION USING UBISOM OUTPUT MODELS FOR CONCERN MINING

The system's design should provide for a continuous incoming stream and storage of MATLAB code files. Analysis and mining of its contents can be done based on ubiquitous data mining algorithms [13]. The UbiSOM algorithm was selected as an illustrative validation of this approach [5]. Appropriate support was developed for the continuous analysis

of data, approached as a data stream in which new m-files and toolboxes can be continuously added. The use of metrics to characterize the relevant blocks of code in the various m-files makes it possible to represent those blocks as a set of measures for different concerns, i.e., a set of feature-value pairs that is also stored in the database in a `patterns` entity. Each new set of such value pairs can be analyzed by means of the UbiSOM algorithm [7]. Note that each concern gives rise to its own specific value for each metric considered.

The UbiSOM component performs a continuous analysis of the data stream and maintains a SOM summarising all m-files in the repository and its contents, updating it whenever new contents are added. The relational model can deal with multiple SOM instances, all of which are represented in the database. This opens the way for (suitably trained) users to specify queries over the repository (using SQL) that also use SOM information to perform selections based on higher-level concepts. Note that each resulting SOM model and related query results can immediately become internally accessible to the system for subsequent processing.

The SOM model used consists of a fixed rectangular grid of units. Each unit can be seen as a generalisation of representations of sets of m-files with similar metric values – also called a *prototype* [14]. In the relational model, the `patterns` entity can also represent the various units of the SOM – again as sets of feature-value pairs. This way, for a given SOM model it is possible to associate each m-file to the SOM unit whose vector of metric values is closer to it (Euclidean distance is used for this purpose). The SOM community would call this unit the *best matching unit* (BMU) for that m-file. Various sub-sets of units in contiguous areas of the SOM, are also aggregated into *regions* of similar units in the SOM, whose information is stored as tuples in the database. Regions and units are represented in the entity `Clusters`.

V. ILLUSTRATIVE QUERIES

This section presents results that can be derived from mining the seed code base for higher level concepts. We call these *intelligent queries*. To facilitate comparison of results, we use an already published set of metrics and corresponding SOM model as an illustrative example [5] and which is copied into the database.

The set of concerns and related metric values are used as different dimensions or positions in the pattern vector fed to UbiSOM. The study refers to two disjoint clusters of m-files that were labelled as regions A_1 and A_3 in the original dataset [6] [5]. We retain the A_1 and A_3 labels in the database mainly to facilitate the task of readers wanting to make the connection with previous work [6] [5]. The description that follows does not depend on details from the other study. Though no m-file can belong to two clusters simultaneously, the previous analysis revealed the simultaneous presence of two or more concerns in those clusters [5]. This is a clear indicator of code tangling, which in turn is a clear indicator of deficient modularity [3] [6] [5]. From this, it can be concluded that the situation in which multiple concerns are found in the



Fig. 2. Code view in the web system.

same m-file arises often. Several such cases are reported in that study [6] [5]. In it, A_1 and A_3 correspond to two SOM regions that represent two disjoint sets of m-files.

One of the concerns detected in m-files from the A_1 and A_3 clusters is *verification of function arguments*. It relates to functions that were prepared to be called in several different "modes", which are selected on the basis of the number of arguments that were passed upon its call. The previous study calls them *schizophrenic functions* [6] [5]. Typically, such functions use the `nargin` function from the MATLAB standard library and/or related functions. `nargin` returns the number of input arguments given in the call. A glimpse of code pattern based on calls to `nargin` is provided in Fig. 2 and Table II. In many cases, these `nargin` calls are made in a considerable number of points at the start of the code's schizophrenic function.

To illustrate the use of queries that join the relational representation of the SOM model with the data in the repository, a query is next shown, which returns the number of tokens associated to a given concern for each m-file covered any of the two regions A_1 and A_3 , i.e., A_1 or A_3 [6] [5]. These regions represent the set of m-files that also give rise to high metric values relative to the concern *verification of function arguments*. Restricting the query to the set of m-files from regions A_1 or A_3 , facilitates the analysis. Note that the restriction could be specified on the basis of *other* concerns (also restricted to A_1 or A_3 in this case). These are an examples of high-level restrictions that would be hard to express without the intelligent assistant for an annotated repository presented in the current paper.

TABLE I
CONTENT (TOP 5 COUNTS) HIGHER LEVEL CONCEPTS QUERY

Cluster	Sub-cluster	m-file	linesCount per m-file	VFAccount per m-file
A_3	19	31424	779	202
A_3	19	12311	480	67
A_3	79	9252	950	49
A_1	739	30854	452	45
A_1	799	24100	162	40

Table I shows an output example of this query. It represents the dataset restricted to clusters A_3 or A_1 (column Cluster), SOM unit identifications (column Sub-cluster), the respective m-file id (column m-file), the number of lines of code of each m-file (column linesCount) and the number of tokens associated to concern *verification of function arguments* (column VFACount). After the query, we learn that the m-file with $id = 31424$ has 202 tokens associated to the concern under analysis (a significantly high value, with 26 occurrences per 100 lines of code). Table II shows a few code lines where token `nargin` occurs, in some cases complemented with the following lines for clarity. It is used in lines 185-189 and again in 195-200. In the second example, `nargin` checks whether the function receives zero arguments. In the third example, an error is issued in case the number of arguments equals 1.

TABLE II
SQL CODE PATTERN '%ARGIN%' IN M-FILE=31324

line	code
1	<code>functionargout = spm_sp(varargin)</code>
	<code>if nargin == 0</code> <code>error('Do what? no arguments given...')</code> <code>else</code> <code>action = varargin{1};</code> <code>end</code>
185-189	<code>otherwise,</code> <code>if nargin==1, error('No space : can't do much!'), end</code> <code>[ok,str] = spm_sp('issetspc',varargin{2});</code> <code>if ~ok, error(str), else, sX = varargin{2}; end;</code> <code>end;</code>
195-200	

The system's current implementation is a web prototype that pays special attention to code visualization. The code view enables token highlighting and provides pop-overs for various kinds of information as illustrated in Fig. 2. A frame is also available for searching *non-contiguous token sequences* in repositories as illustrated in Fig. 3. It also provides a *TreeMap* view (not shown) suitable for the visualisation of hierarchical data, e.g., *toolboxes > m-files > concern* [15], [16]. A visualization of the SOM model is also provided, which allows the selection of elements to derive a database relation of m-files belonging to specific SOM units and their regions.

VI. DISCUSSION

One may ask why a relational model was used instead of say a noSQL-type model. Granted, noSQL-type and graph databases may offer specific advantages in some cases. However, we opted for a relational model because token-based data comprises a significantly structured domain. The relational model facilitates integration with additional components and transaction support to cope with the addition of new annotations on the part of different users. It also facilitates efficiency gains for some queries, which are left for future work.

Mathworks¹ maintains the MATLAB Central website, which aims to provide the best support for MATLAB. It

¹The leading company proprietary and developing MATLAB language.

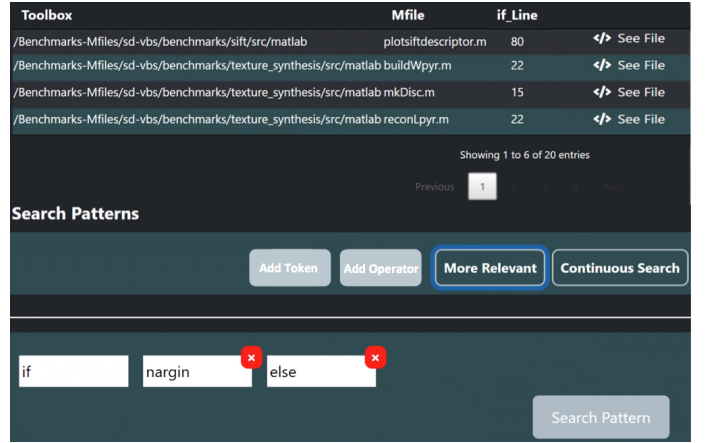


Fig. 3. Example of the web interface frame for token sequences.

has a forum that is claimed to contain 110,000 answered questions and a *File Exchange toolbox repository* for sharing code, where developers can easily import their toolboxes from GitHub [17]. As in every other software repository that we are aware of, the focus is on the toolbox. All toolboxes can be tagged and some of those toolboxes have online tutorials or even Webinars. There is also an area for MATLAB code examples, conveniently indexed by main topics in the language (e.g. matrixes and arrays) and highlighting to toolsets with example code. Each function also points to code examples. Unfortunately, MATLAB developers do not seem to have direct query access to that huge repository of code. We searched many repositories in several other major programming languages but failed to find a system managing a repository of MATLAB programs and toolboxes that supports global analyses and enables extraction of higher-level concepts. By contrast, this paper proposes a software repository enabling searches down to the level of tokens and which are still able to link results to the enclosing toolbox.

Tokens are usually defined as the smallest individual elements of any program. Queries over tokens allow the search of all the information in the repository. As token usages are also very diverse and case specific, there is a need for higher level searches, namely over concerns. However, there are too many possible types of tokens and token combinations. The use of concerns in queries allow for a more direct representation of the concepts involved in the reasoning of software developers when working on the code, thus bridging an important conceptual gap. To search for concerns, we need to resort to their manifestation in the code, by means of metrics on code patterns [5]. The present work is based on *concern metrics* and also relates to past research work on that category of metrics [18] [19].

It should be noted that the higher level concepts used in the illustrative queries are the combinations of concerns used in section V, which refers to our recent work on this front [5]. Presently, the database contains just the clusters and regions explained in that model [5]. Searches over the higher

level concepts with combinations of concerns proved very promising for several kinds of search regarding the concerns present in the code. However, the database and supporting site are more general. It is a relevant subject for this research to find other UbiSOM models (possibly with new metrics) that may answer different questions. The system is general enough to frame and annotate results from new knowledge discovery models and we are available to collaborate with the community to add new models to the system.

Past work used the UbiSOM algorithm to explore the related set of concern metrics [5], through which manual analyses of SOM results led to the detection of many cases of the joint occurrence of multiple concerns in a single m-file. However that method was still based on a generic data mining tool and unable to query a software repository. The repository here described uses the SOM as a data-mining tool to identify clusters of m-files having similar patterns regarding possible distinct sets of concern metrics. The SQL queries that can be devised over such clusters provide the advanced user with higher level concepts. Such queries result in sets of m-files available in the repository. A visual web interface allows end-users (namely MATLAB programmers) to search over relevant higher level concepts, such as the illustrative *schizophrenic functions* concept described in V.

VII. CONCLUSION

This paper proposes a software repository management system supporting intelligent queries over MATLAB code files and able to associate them to higher level concepts. This is achieved by the synergistic combination of a token extraction tool, a relational database and the advanced exploratory capabilities of a Self-Organizing Map. A web interface supports queries over the resulting knowledge stored in a relational model. The latter supports a token-based advanced exploration of the repository. Higher level concepts from SOMs – based on software concerns – can be used by programmers, by means of the web interface. A demonstration web site with full illustrative examples of such higher level concepts and supplementary material is available ². In addition, direct access to the repository database is freely available for research purposes.

Regarding future work, SOMs can be used for tackling problems other than those covered in this paper. SOM models are already available in our database, which opens the way for extending the present annotation with additional data mining processes. For instance, WEBSOM [20] is a classical use of SOM to document clustering. Line comments provide another interesting opportunity: if we approach them as documents, the joint inclusion of such a SOM in our database comprises a promising topic for future work on mining.

REFERENCES

- [1] J. M. Cardoso, J. M. Fernandes, and M. P. Monteiro, "Adding aspect-oriented features to matlab," in *Fifth International Conference on Aspect-Oriented Software Development (AOSD 2016)*, 2006.
- [2] T. Aslam, J. Doherty, A. Dubrau, and L. Hendren, "Aspectmatlab: An aspect-oriented scientific programming language," in *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, pp. 181–192, ACM, 2010.
- [3] M. Monteiro, J. Cardoso, and S. Posea, "Identification and characterization of crosscutting concerns in matlab systems," in *Conference on Compilers, Programming Languages, Related Technologies and Applications (CoRTA 2010)*, Braga, Portugal, pp. 9–10, 2010.
- [4] J. M. Cardoso, J. M. Fernandes, M. P. Monteiro, T. Carvalho, and R. Nobre, "Enriching matlab with aspect-oriented features for developing embedded systems," *Journal of Systems Architecture*, vol. 59, no. 7, pp. 412–428, 2013.
- [5] N. Cavalheiro Marques, M. Monteiro, and B. Silva, "Analysis of a token density metric for concern detection in matlab sources using ubisom," *Expert Systems*, vol. 35, no. 4, 2018.
- [6] M. P. Monteiro, N. C. Marques, B. Silva, B. Palma, and J. Cardoso, "Toward a token-based approach to concern detection in matlab sources," in *proceedings of the 18th Portuguese Conference on Artificial Intelligence*, pp. 573–584, Springer, 2017.
- [7] B. Silva, *Exploratory Cluster Analysis from Ubiquitous Data Streams using Self-Organizing Maps*. PhD thesis, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 12 2016. Manuscript available at: <http://hdl.handle.net/10362/19974>.
- [8] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton Jr, "N degrees of separation: multi-dimensional separation of concerns," in *Proceedings of the 21st international conference on Software engineering*, pp. 107–119, ACM, 1999.
- [9] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Longtier, and J. Irwin, "Aspect-oriented programming," in *Proceedings of 11th European Conference on Object-Oriented Programming*, pp. 220–242, Springer, 1997.
- [10] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, pp. 377–387, June 1970.
- [11] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database system concepts*. New York: McGraw-Hill, 6 ed., 2010.
- [12] J. Bispo and J. M. P. Cardoso, "A matlab subset to c compiler targeting embedded systems," *Software: Practice and Experience*, vol. 47, no. 2, pp. 249–272, 2017.
- [13] J. Gama, *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, 1st ed., 2010.
- [14] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.
- [15] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pp. 214–223, ACM, 2005.
- [16] B. Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," *ACM Transactions on graphics (TOG)*, vol. 11, no. 1, pp. 92–99, 1992.
- [17] "Math Works matlab central website." <https://www.mathworks.com/matlabcentral/>. Accessed: 2019-01-22.
- [18] E. Figueiredo, C. Sant'Anna, A. Garcia, T. T. Bartolomei, W. Cazzola, and A. Marchetto, "On the maintainability of aspect-oriented software: A concern-oriented measurement framework," in *Proceedings of the 12th European Conference on Software Maintenance and Reengineering*, pp. 183–192, IEEE, 2008.
- [19] E. Figueiredo, C. Sant'Anna, A. Garcia, and C. Lucena, "Applying and evaluating concern-sensitive design heuristics," *Journal of Systems and Software*, vol. 85, no. 2, pp. 227–243, 2012.
- [20] S. Kaski, T. Honkela, K. Lagus, and T. Kohonen, "Websom—self-organizing maps of document collections1," *Neurocomputing*, vol. 21, no. 1-3, pp. 101–117, 1998.

²<http://bit.ly/MatlabAnnotatedRepository>

BDFIS: Binary Decision Access Control Model Based On Fuzzy Inference Systems

Diogo Domingues Regateiro¹, Óscar Mortágua Pereira², Rui L. Aguiar³

Instituto de Telecomunicações
DETI, University of Aveiro
Aveiro, Portugal
{diogoregateiro¹, omp², ruilaa³}@ua.pt

Abstract—Access control is a ubiquitous feature in almost all computer systems, and as data becomes more and more of an important asset for organizations, so do the associated access control policies. However, with the increase in the amount of data being produced, e.g. in IoT and social networks, the interest in simpler access control is increasing as well since more subjects (public, researchers, etc.) are now requesting access to it. Defining the exact conditions to allow each subject to access the data can be difficult, especially when vaguely defined conditions such as "expertise of a researcher" come into play. Fuzzy Inference Systems (FIS) allow to process these vague conditions and enables access control mechanisms to be more easily applied. The contribution of this paper lies in showing how a FIS can be used to output binary access control decisions (grant/deny) and what are the differences in the inference process that stems from restricting the output to these two output values.

Keywords—fuzzy systems, vague knowledge, information security, access control.

I. INTRODUCTION

Access control has always been an important feature in any system, be it physical or digital, as it restricts access to a resource in a controlled and selective manner [1]. The most successful access control models are usually those that mimic real-world ways of managing permissions within the context of their application, of which the Role-based Access Control (RBAC) [2] model is a key reference. RBAC is a classical model that maps subjects trying to access some resource to a role, a meaningful category within the context of the system being protected, and crisp access control rules define the resources a subject playing a given role may access to successfully complete its tasks. Other classical access control models operate in a similar manner, using crisp rules that clearly define which resources each subject may access.

However, with the advent of big data and social networks, the quantity and complexity of data available that needs to be stored and processed have increased considerably. Classical access control models are ill-suited to handle these scenarios, as they require tight mappings between subjects, objects, and permissions. This means new subjects must be manually assigned to their permissions before they can access the data, which introduces delays and adds security management loads.

Additionally, the real world is not always as unambiguous as

the classical access control models require it to be in their policies. For example, some documentation from European projects may not be publicly available but could be disclosed to experts researching in some area related to a project. There is no hard definition of what makes someone an expert, so normally it would have to be checked manually on a case-by-case basis. In such situations, the fuzzy set theory is an appropriate solution since it can handle vague concepts, such as expertise of the subject, without requiring crisp values within its rules. Thus, allowing policy rules to be richer in meaning and flexibility.

The theory of fuzzy logic [3] aims to capture how human perception and cognition interpret the world, which is not unambiguous all the time. To this end, it uses relative graded memberships between a subject and a vague concept. Thus, fuzzy logic permits the inclusion of vague human assessments in computing problems, which proved to be an effective way to deal with multi-criteria problems [4]. Solutions known as fuzzy inference systems (FIS) were then designed to map a set of inputs to outputs, using fuzzy logic and fuzzy sets to define vague conditions. These characteristics allowed fuzzy logic and fuzzy set theory to find a lot of uses in various areas, such as medicine [5]–[7], computer security [8]–[14], networking [15], [16], aeronautics [17], stock trading [18], and many others [4], [19]–[22].

Consider a community managed public data, like a wiki, where only subjects that are experts (to some degree) on the contents of a page would be able to modify it. Since wiki pages are generally given categories related to their contents and other related tags, the access control system could access services such as Scopus to retrieve the number of publications, their keywords, citations, etc. This information could then be fed into a fuzzy inference system to determine the level of expertise of the user and help the access control system to make its access control decisions. Thus, leading to fewer modifications by users with malicious intents and more modifications with quality content.

Thus, the contribution of this paper lies in proposing an access control model that uses a FIS to make binary access control decisions, herein known as BDFIS; the presentation of application scenarios where such a system could be useful; and what are its benefits/issues when compared to other access control mechanisms.

The rest of this paper is organized as follows: section II will

This work is funded by FCT/MEC through national funds and when applicable co-funded by FEDER – PT2020 partnership agreement under the project UID/EEA/50008/2019 and SFRH/BD/109911/2015.

provide some of the state of the art in regards to the application of fuzzy set theory in access control; section III will provide the analysis made to each step made during the output generation process of the BDFIS; section IV will present a proof of concept of the proposed model; and section V will provide a short discussion while addressing the issues found.

II. RELATED WORK

The fuzzy set theory is a topic that has been researched in recent years to tackle scenarios where the information that needs to be processed is vague, which can include management science, politics, social psychology, artificial intelligence, and access control, among others [4]. This capability to handle vague information is what enables it to be useful in scenarios where binary decisions must be made and the decision rules are difficult to define in a crisp manner.

Surprisingly, it was found that there is very little research done in terms of the application of fuzzy set theory in access control systems. The issue is suspected to come from the fact that by using vague conditions in the form of fuzzy sets, a fuzzy-based access control system does not explicitly state which input values would grant access to some resource which would not. This paper is focused on exploring these limitations and where fuzzy systems can be improved for this area of intervention.

In [23], the authors introduce Fuzzy Role-based Access Control (FRBAC), which uses fuzzy relations between users-roles and roles-permissions:

- $USERS \times ROLES \rightarrow [0, 1]$
- $ROLES \times PERMISSIONS \rightarrow [0, 1]$

This approach allows for users to have partial permission assignments, which are then used to calculate the access degree they have to each resource. Then, if the access degree is used directly to control access to a resource, the resource itself must have fractional access, defined using the following access function where $USERS$ is the set of users, OPS the set of operations and OBS the set of objects:

- access: $USERS \times OPS \times OBS \rightarrow [0, 1]$

Martínez-García et al. define a function that takes a threshold variable δ (i.e. a value between 0 and 1) and returns *grant* if the access degree is greater than δ or *deny* if not. However, this approach still restricts the fuzzy sets to roles. Therefore, it limits the type of access control logic that can be used. BDFIS, in contrast, does not require subjects to be mapped to the protected resources through any specific model, such as roles, allowing to abstract any mapping between them.

Another work was found where the trust level of devices is measured, so a fuzzy approach to trust-based access control could be achieved (FTBAC) [11]. This is done by capturing information about the devices to determine the vague concepts *Experience (EX)*, *Knowledge (KN)* and *Recommendation (RC)*, and several fuzzy sets (linguistic terms) were defined for each one. The following values for each concept are calculated for a context c between two devices A and B , used as inputs for the membership functions of the linguistic terms.

EX depends on the history of interactions v_k between A and

B , where $k \in [0, n]$, incrementing or decrementing when a positive or negative interaction occurs, as shown in (1).

$$(EX)^c = \frac{\sum_{k=1}^n v_k}{\sum_{k=1}^n |v_k|} \quad (1)$$

KN is calculated with the help of direct knowledge d , indirect knowledge r , and their respective weights (W_d, W_r), where $d, r \in [-1, 1]$, $W_d, W_r \in [0, 1]$, and $W_d + W_r = 1$, as shown in (2).

$$(KN)^c = W_d * d + W_r * r \quad (2)$$

The RC is calculated by device A based on the summation of the RC values from n other devices about device B . W_i and $(r_c)_i$ are weights assigned by device A to the recommendation of the i^{th} device and its RC value respectively, where $r_c \in [-1, 1]$ and $W_i \in [0, 1]$, as shown in (3).

$$(RC)^c = \frac{\sum_{i=1}^n W_i * (r_c)_i}{\sum_{i=1}^n (r_c)_i} \quad (3)$$

Different permissions can be mapped to different levels of trust, so depending on the level of trust the granted permissions change. This the access decisions solely based on the level of trust. If there are other access conditions, they need to be considered separately. Since BDFIS can abstract any mapping rule between subjects and resources, the level of trust can be used in the same manner. However, unlike FTBAC, other access requirements can be added without issue to the inference system.

Another work was carried out that uses fuzzy set theory to calculate a measure of risk and applies it to enhance the access security of eHealth cloud applications [10]. To achieve this, three different inputs are used: *data sensitivity*; *action severity*; and *risk history*. Next, a set of rules is applied to calculate the level of risk associated. A crisp output value is then determined by applying a defuzzification technique, which indicates the overall level of risk as a percentage. However, the process to determine whether the access should be granted given a risk level is not detailed. Moreover, this approach is specific to the measurement of the level of risk with a given access attempt. This limits the applicability of this approach when compared to BDFIS, which can use most concepts in its policies.

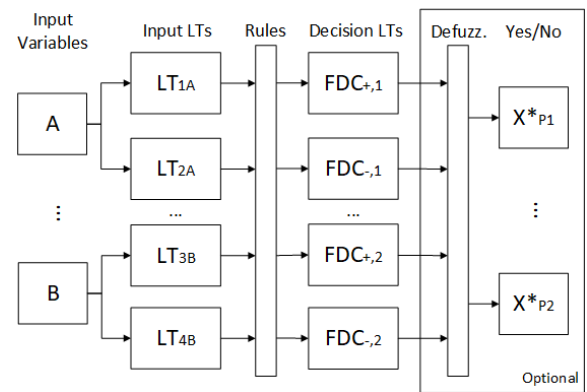


Figure 1. Conceptual BDFIS block diagram.

III. BINARY DECISION FIS ANALYSIS

In this section, a FIS is analyzed regarding its applicability to binary decision making (access control grant/deny decisions) in each of its processing steps.

There are many different types of FIS [24]–[28]. However, the Mamdani-type FIS [28] was chosen for analysis since it is a type of system that is commonly available on most FIS implementation tools, has widespread use and it was found to be easily adaptable to support binary decisions. The Sugano-type FIS [27] is also commonly available in such tools, but it falls short as it has less expressive power and interpretability than the Mamdani-type FIS [29]. Fig. (1) shows the conceptual BDFIS that will emerge from the analysis made in this section and will serve as an illustrative guide to the proposed modifications.

The standard Mamdani-type FIS goes through the following set of steps during processing:

1. The determination of the set of fuzzy rules by an expert in the application context (i.e. the rules block);
2. The fuzzification of the input variable values into the input linguistic terms (LTs) using the associated membership functions;
3. The application of the fuzzy rules to establish the rule strengths to the output LTs, known as the fuzzy decision components (FDC) in the BDFIS;
4. The combination of the rule strength and the output LTs membership functions to determine the consequence functions for each output variable;
5. The combination of the consequence functions to get an output distribution function for each output variable;
6. The defuzzification step, which outputs a single crisp value for each output variable (a decision in the context of the BDFIS) given an output distribution function (required only if a crisp output is needed).

These steps will be detailed in the following subsections and how they were modified for the BDFIS.

A. Fuzzy Rule Determination

The fuzzy rule determination process involves deciding which linguistic terms are going to be used within the FIS, both for input and output variables, and how they influence each other using predefined rules.

The input linguistic terms should stay effectively the same as they still qualify the attributes available in the application context. The input linguistic terms and their membership functions are still required to be written by an expert.

The output variables and linguistic terms, however, are dependent on the decisions must be made. In the case of an access control system, the decisions are either to grant or deny a subject some permission to a resource. Thus, permissions can be declared as the output variables according to Def. (1).

Definition 1. Access permissions in a BDFIS are output variables associated with exactly two FDC linguistic terms: one for a positive decision FDC_+ (yes/grant); and one for a negative decision FDC_- (no/deny).

The rules can then take input linguistic terms from one or more input variables and establish a relation to one of the FDCs. To illustrate, consider two input variables A and B , with the linguistic terms LT_A and LT_B , and an output variable Z . A rule can take the form "if A is LT_A and/or B is LT_B then Z is FDC_{\pm} ". For example, "if *Expertise* is *High* and *Activity* is *Moderate* then *Read* is *Granted*."

This shows how a FIS can be used to easily encode vague access conditions: given a set of vague concepts about a subject (e.g. *Expertise*, level of *Activity*, etc.), the permissions (*Read*, *Write*, etc.) to the resource are output variables that are defined by either being granted (FDC_+) or denied (FDC_-). Furthermore, the permission and decision pair are easily identifiable.

B. Input Fuzzification And Rule Strength

The input fuzzification process and rule strength determination are steps that qualify the input variables in terms of the defined linguistic terms. Given a set of linguistic terms T_i for an input variable i , the membership degree of a subject s to each linguistic term $t \in T_i$ is obtained by applying that linguistic term membership function μ_t , as shown in (4).

$$\mu_t(s): t \rightarrow [0,1], t \in T_i \quad (4)$$

To illustrate, if $\mu_t(x) = x/20, 0 \leq x \leq 20$ is used to define the "high" linguistic term for the "number of publications" input variable, then if a subject has 15 publications it has a membership degree of $15/20 = 0.75$ to that linguistic term. The membership functions μ are defined by an expert in the application context the BDFIS is to be deployed on, since vague concepts like *Expertise* can change slightly depending on the context. After a membership degree is calculated for each input linguistic term, the rule strength for each output FDC can be determined. This is usually done by applying the fuzzy logic operators as dictated by the rules (AND, OR, and NOT). If more than one rule applies to the same FDC, the rule strength of each such rule is unified by typically applying the OR operator. These operators have several different implementations for fuzzy logic that can be used, but they always satisfy the De Morgan's Laws.

To reiterate, there are only two possible output linguistic terms: the FDC_- and the FDC_+ . This makes it clear for which outcome a rule is being used for instead of having something more abstract, such as the user expertise level, and lets the system make access control decisions based on it.

C. Consequence Determination

The next step is to determine the consequence of the rules and to do so it is necessary to think about what the output is intended to be.

The goal is to have a FIS that can make a binary access control decision for each permission (i.e. grant or deny). As such, each decision is defined by two linguistic terms, the FDC_+ and FDC_- output linguistic terms previously introduced. These represent the positive and negative decisions for a single output, respectively. This way, a subject attempting to access some piece of information or service is mapped automatically through rule strengths to each FDC.

Since each FDC is also a fuzzy set, each can have any

membership function that the security expert chooses. However, a simpler approach is proposed to reduce the complexity of the calculations and the potential performance bottleneck that running a FIS can introduce. Instead of a user-defined output membership function, each FDC will have a predefined singleton function (see Def. (2)) instead.

Definition 2. A given function $f(x)$ is a singleton function if its output is always 0 except for a single input value x_0 , for which its output is 1 as shown in (5).

$$f(x) = \begin{cases} 1, & \text{if } x = x_0 \\ 0, & \text{if } x \neq x_0 \end{cases} \quad (5)$$

Since it is expected for the BDFIS to output a decision, it makes sense that they are each associated with a single value. This is partially the reason why the membership function for each FDC is proposed to be a singleton function. This is done by setting the single function x_0 value to 0 for the FDC_- and to the value 1 for the FDC_+ . These values were carefully chosen since they simplify the defuzzification step considerably, which will be shown in section III.D.

Thus, the membership functions of the FDC_- ($\mu_-(x)$) and FDC_+ ($\mu_+(x)$) are singletons that are defined as shown in (6) and (7), respectively.

$$\mu_-(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{if } x \neq 0 \end{cases} \quad (6)$$

$$\mu_+(x) = \begin{cases} 1, & \text{if } x = 1 \\ 0, & \text{if } x \neq 1 \end{cases} \quad (7)$$

Finally, the process of determining the consequence consists of truncating the output membership function, i.e. both $\mu_-(x)$ and $\mu_+(x)$. Def. (3) shows how this process is accomplished.

Definition 3. The process of truncating a given function f at the value $y = y_0$ generates a new function g that has the same output as f except that any output value greater than y_0 becomes y_0 , as shown in (8).

$$g(x) = \min(f(x), y_0) \quad (8)$$

Determining the consequence from these singleton functions also becomes easier for the following reasons:

1. Singleton functions only have one input value ($x = x_0$) for which the output is not 0, thus only one value may have to be truncated;
2. The FDC singleton membership functions always output 1 for the value $x = x_0$;
3. The rule strength applicable to each FDC always lies within the range $[0,1]$ since it is the result of the application of fuzzy logic.

Since the output membership function of each FDC is either 0 or 1 as stated in reasons (1) and (2) and the rule strength is a value within the range $[0,1]$ as stated in reason (3), the consequence function C is determined by simply replacing the output value 1 with the rule strength RS for that FDC , as shown

in (9) for FDC_- and (10) for FDC_+ .

$$C_-(x) = \begin{cases} RS_-, & \text{if } x = 0 \\ 0, & \text{if } x \neq 0 \end{cases} \quad (9)$$

$$C_+(x) = \begin{cases} RS_+, & \text{if } x = 1 \\ 0, & \text{if } x \neq 1 \end{cases} \quad (10)$$

These consequence functions can then be used in the defuzzification step to generate a single, crisp output value. This explains in part how using singleton functions simplifies the computation of the final decisions. However, further benefits from this approach will be explored in the following subsection.

D. Consequence Combination And Defuzzification

The next step in the process involves combining the consequence functions of both FDCs (C_- and C_+) into an output distribution function for each output decision. This allows to apply a defuzzification method, an inverse transformation to the fuzzification step, that outputs a crisp output value for each output decision.

These steps are optional and depend on the level of information the system requires to reach a decision. If the system requires more than a crisp output value, it can use the rule strengths given to each FDC and make a more informed decision this way. For example, a use case could require an access control system to ask a human to manually grant or deny access if the rule strengths of both FDCs are close to one another.

However, if the system requires a crisp value between 0 and 1, then the consequence functions can be combined into an output distribution function (Def. (4)) and a defuzzification step may be used.

Definition 4. Given the consequence functions C_- and C_+ of an output variable O , the output distribution function θ associated with O is the result of applying an accumulative function S , as shown in (11).

$$\theta(x) = S(C_-, C_+) \quad (11)$$

The approach used in this paper uses the maximum accumulative function. Consider that both FDC_- and FDC_+ have RS_- and RS_+ rule strengths respectively. The resulting output distribution function for each output decision Z (θ_Z) is shown in (12).

$$\theta_Z(x) = \begin{cases} RS_-, & \text{if } x = 0 \\ RS_+, & \text{if } x = 1 \\ 0, & \text{if } x \neq 0 \wedge x \neq 1 \end{cases} \quad (12)$$

Therefore, all output distribution functions are the combination of two singleton functions, one for the FDC_- on the x value 0 and the other for the FDC_+ on the x value 1.

A defuzzification method can then be applied to each output distribution function generated this way for each output variable. To show why the selected x values for each singleton were chosen, the commonly used center of gravity for singletons (COGS) defuzzification technique will be applied to θ_Z . The general COGS formula for a given output distribution function

θ_z is given in (13).

$$\text{COGS}(\theta_z) = \frac{\sum_x x * \theta_z(x)}{\sum_x \theta_z(x)} \quad (13)$$

Note that if the output distribution function θ_z is not generated from singleton membership functions then the formula for the center of gravity is a division of two primitives. Fortunately, since the proposed output distribution functions are always the combination of two singletons, which are also always defined on the x values 0 and 1, the formula (14) follows:

$$\text{COGS}(\theta_z) = \frac{0 * \theta_z(0) + 1 * \theta_z(1)}{\theta_z(0) + \theta_z(1)} = \frac{\theta_z(1)}{\theta_z(0) + \theta_z(1)} \quad (14)$$

As it can be seen, the COGS formula was simplified to a simple fraction of the rule strength of the FDC_+ to the sum of the rule strengths of the FDC_- and FDC_+ , reducing its complexity.

The application of the COGS defuzzification method to each output distribution function results in a crisp output value, which can be used to arrive at a final decision. The simplest way to achieve this is to set a fixed threshold, such as 0.5, and if the crisp output is lower than the threshold then the decision is negative (*deny*, in the access control context), otherwise, it is positive (*grant*, in the access control context). The threshold value can be increased or decreased to fine-tune the system as required by an expert. Any other method to arrive at a decision is valid, such as the maximum method which takes the x value that maximizes the output distribution function and depends only on the use case.

E. BDFIS Analysis

A modified Mamdani-type FIS called BDFIS has been proposed and detailed in this paper. However, upon further analysis, it was found that the consequence determination lacks some of its former expressibility. The reason behind this comes from the fact that the output linguistic terms and distribution functions are now fixed, i.e. FDCs and singleton functions respectively, while these functions could be defined freely in a standard FIS. This forces the rule strengths for the FDCs to be calculated directly from the input linguistic terms, where abstract concepts such as *Expertise* could be defined in a standard FIS instead. Thus, the ability to define abstract concepts is hindered to some degree.

While this fact may not impact use cases with simpler access control policies, it can impact the interpretability of the defined rules in others. Consider a rule that defines the vague concept of *Expertise*, such as “**if** *Number_of_Publications* is *High* **then** *Expertise* is *High*”. If someone is newly hired to manage a system that uses this rule, it is clear what it is expressing: the expertise of the subject.

As is, the BDFIS would need the input variables to be mapped directly to the output decision variables, meaning that the *Expertise* vague concept cannot be explicitly defined or used in the mapping to the FDCs. Thus, a new security expert would need more detailed external documentation to understand what the rules are meant to represent to master the system, to write new rules, etc. However, it is possible to add a second layer of rules and intermediate variables to allow this (or more for additional abstraction). The input linguistic terms are mapped to

these intermediate variables, which can include vague concepts like *Expertise*, and then these variables are mapped to the FDCs. This approach allows for rules to remain easily interpretable by humans at the cost of some processing.

IV. PROOF OF CONCEPT

In this section, a proof of concept of the BDFIS will be shown. The prototype of the BDFIS used for this proof of concept uses JFuzzyLogic[30][31] and is available at github.com/Regateiro/FuzzyAC/tree/master/java/BDFIS. The *academic.fcl* file defines a BDFIS that makes access control decisions based on the expertise of a subject, using the Fuzzy Control Language (FCL)[32]. The BDFIS also comprises of two blocks of variables and rules, the first calculates the degree of expertise of the subject and the second the access control permissions. Fig. (2) shows the output of the proof of concept BDFIS implementation using the provided FCL file.

```

1 | -- INPUT: Number_Of_Citations (50.000000)
2 | | -- TERM: High (0.285714)
3 | | -- TERM: Low (0.000000)
4 | | -- TERM: Considerable (0.750000)
5 | ...
6 | -- OUTPUT: Expertise (2.648649)
7 | | -- TERM: High (0.285714)
8 | | -- TERM: Low (0.000000)
9 | | -- TERM: Medium (0.750000)
10 | | -- TERM: Very_High (0.285714)
11 | ...
12 | -- OUTPUT: Read (1.000000)
13 | | -- TERM: Grant (0.750000)
14 | | -- TERM: Deny (0.000000)
15 | |
16 | -- OUTPUT: Write (0.275862)
17 | | -- TERM: Grant (0.285714)
18 | | -- TERM: Deny (0.750000)
19 |
20 | ***** RESULTS *****
21 | [Read] permission is GRANTED.
22 | [Write] permission is DENIED.

```

Figure 2. Sample BDFIS implementation output.

The defined BDFIS takes two input variables: the *number of publications* (omitted); and the *number of citations* (lines 1-4). These are used to determine the *Expertise* of the subject in the first set of rules (lines 6-10). The *Expertise* is then applied in the second set of rules to calculate the FDCs for a *Read* and *Write* permissions (lines 12-18). These are necessary to calculate whether the subject is granted each permission, by checking if the final defuzzified value is greater than 0.5 (grant) or not (deny). This value can be modified by an expert to require subjects to have lower or higher membership degrees to be granted access.

The numbers to the right of the input and output variables denote the crisp value associated with them (either provided as input or calculated from defuzzification). The number to the right of the terms denote their associated membership degrees. These are calculated from piecewise linear membership functions and rules defined by an expert in the *academic.fcl* file using the steps and methods explained in this paper. The BDFIS implementation automatically passes the *Expertise* value calculated by the first set of rules to the second set of rules as an input. Finally, since the *Read* permission has the output value $1.0 > 0.5$, it is granted, and the *Write* permission with the output value $0.275862 < 0.5$ is denied.

V. DISCUSSION

In this paper, a FIS that can make binary access control decisions was proposed. Through the analysis made to each processing step, several changes were introduced that allowed a Mamdani-type FIS to specialize in the output of binary decisions. Furthermore, some proposed modifications optimized the output generation process, making this type of systems to have the potential for deployment in access control scenarios that deal with vague knowledge. Given the broad spectrum of areas where FIS are used, it shows that they are useful if correctly defined. By creating the BDFIS as closely as possible to a standard FIS, it is expected that it can be just as effective in access control scenarios.

One might question the appropriateness of applying fuzzy set theory to model access control since its inherent vagueness prevents an expert from easily knowing if a subject can access the protected data or not. Furthermore, auditing such a system is not easy given that new subjects may request access at any time.

Due to the vague nature of the fuzzy access control rules, the exact ranges of input values that grant access to the data is not clear. Thus, the possibility that an unexpected combination of input values granting access to the data may exist. It is important to note, however, that such a fuzzy access control model would still be deterministic. Furthermore, since vague conditions are applicable to a range of input values (with varying membership degrees), fewer rules are needed when compared to classic models, which generally requires each combination of input values to be written down as a separate rule. Nonetheless, fuzzy-based access control models are not easily accepted to manage sensitive data (e.g. hospital patient data), understandably due to the issues related to its inherent vagueness.

For future work, it is intended to build a system that can audit a BDFIS for correctness, an important feature for any access control system to have, and the calibration of the threshold values. Since the subject parameters used as input values are fuzzified, it is hard to determine from the fuzzy rules if they have access to a resource or not. Thus, being able to determine which input values grant or deny access is an important step towards making BDFIS a viable part of an access control system.

REFERENCES

- [1] R. Shirey, "Internet Security Glossary, Version 2," Aug. 2007.
- [2] D. Ferraiolo and D. Kuhn, "Role-based access controls," *15th Natl. Comput. Secur. Conf.*, 1992, pp. 554–563.
- [3] L. a. Zadeh, "Fuzzy sets," *Inf. Control*, vol. 8, no. 3, 1965, pp. 338–353.
- [4] H. Singh *et al.*, "Real-Life Applications of Fuzzy Logic," *Adv. Fuzzy Syst.*, vol. 2013, 2013, pp. 1–3.
- [5] N. H. Phuong and V. Kreinovich, "Fuzzy logic and its applications in medicine," *Int. J. Med. Inform.*, vol. 62, no. 2–3, Jul. 2001, pp. 165–173.
- [6] M. A. Ghahazi, M. H. Fazel Zarandi, M. H. Harirchian, and S. R. Damirchi-Darasi, "Fuzzy rule based expert system for diagnosis of multiple sclerosis," in *2014 IEEE Conference on Norbert Wiener in the 21st Century (21CW)*, 2014, pp. 1–5.
- [7] S. SushilSikchi, S. Sikchi, and A. M. S., "Fuzzy Expert Systems (FES) for Medical Diagnosis," *Int. J. Comput. Appl.*, vol. 63, no. 11, Feb. 2013, pp. 7–16.
- [8] S. Berenjian, M. Shajari, N. Farshid, and M. Hatamian, "Intelligent Automated Intrusion Response System based on fuzzy decision making and risk assessment," in *2016 IEEE 8th International Conference on Intelligent Systems (IS)*, 2016, pp. 709–714.
- [9] S. Al Amro, F. Chiclana, and D. A. Elizondo, "Application of Fuzzy Logic in Computer Security and Forensics," in *Studies in Computational Intelligence*, vol. 394, 2012, pp. 35–49.
- [10] J. Li, Y. Bai, and N. Zaman, "A Fuzzy Modeling Approach for Risk-Based Access Control in eHealth Cloud," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2013, pp. 17–23.
- [11] P. N. Mahalle, P. A. Thakre, N. R. Prasad, and R. Prasad, "A fuzzy approach to trust based access control in internet of things," in *Wireless VITAE 2013*, 2013, pp. 1–5.
- [12] H. Takabi, M. Amini, and R. Jalili, "Enhancing Role-Based Access Control Model through Fuzzy Relations," in *Third International Symposium on Information Assurance and Security*, 2007, no. 500, pp. 131–136.
- [13] M. Botha and R. von Solms, "Utilising fuzzy logic and trend analysis for effective intrusion detection," *Comput. Secur.*, vol. 22, no. 5, 2003, pp. 423–434.
- [14] V. C. V. Hu, D. F. Ferraiolo, and D. R. Kuhn, "Assessment of access control systems," *Nistir 7316*, 2006, p. 60.
- [15] V. Henn, "Fuzzy route choice model for traffic assignment," *Fuzzy Sets Syst.*, vol. 116, no. 1, Nov. 2000, pp. 77–101.
- [16] S. N. Shiaeles, V. Katos, A. S. Karakos, and B. K. Papadopoulos, "Real time DDoS detection using fuzzy estimators," *Comput. Secur.*, vol. 31, no. 6, 2012, pp. 782–790.
- [17] J. Luo and E. Lan, "Fuzzy Logic Controllers for Aircraft Flight Control," in *Fuzzy Logic and Intelligent Systems*, vol. 3, Dordrecht: Springer Netherlands, 1995, pp. 85–124.
- [18] S. Othman and E. Schneider, "Decision making using fuzzy logic for stock trading," in *2010 International Symposium on Information Technology*, 2010, pp. 880–884.
- [19] Ying Bai, H. Zhuang, and D. Wang, *Advanced Fuzzy Logic Technologies in Industrial Applications*. London: Springer London, 2006.
- [20] W. Liu and H. Liao, "A Bibliometric Analysis of Fuzzy Decision Research During 1970–2015," *Int. J. Fuzzy Syst.*, vol. 19, no. 1, Feb. 2017, pp. 1–14.
- [21] Y. Wardhana, B. Hardian, G. Guarddin, and H. Rasyidi, "Context aware door access control on private room using fuzzy logic: Case study of smart home," in *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 2013, pp. 155–159.
- [22] A. C. F. Guimarães and C. M. F. Lapa, "Fuzzy inference to risk assessment on nuclear engineering systems," *Appl. Soft Comput.*, vol. 7, no. 1, Jan. 2007, pp. 17–28.
- [23] C. Martínez-García, G. Navarro-Arribas, and J. Borrell, "Fuzzy Role-Based Access Control," *Inf. Process. Lett.*, vol. 111, no. 10, 2011, pp. 483–487.
- [24] F. Qian, S. Sen, and O. Spatscheck, "[JJ]Characterizing resource usage for mobile web browsing," *MobiSys '14*, 2014, pp. 218–231.
- [25] R. Werneck, J. Setubal, and A. da Conceição, "(old) Finding minimum congestion spanning trees," *J. Exp. Algorithmics*, vol. 5, 2000, p. 11.
- [26] M. Conti, R. Di Pietro, L. V Mancini, and A. Mei, "(old) Distributed data source verification in wireless sensor networks," *Inf. Fusion*, vol. 10, no. 4, 2009, pp. 342–353.
- [27] M. Sugeno, "Industrial applications of fuzzy control," *Elsevier Sci. Pub. Co.*, 1985.
- [28] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *Int. J. Man. Mach. Stud.*, vol. 7, no. 1, 1975, pp. 1–13.
- [29] A. Kaur and A. Kaur, "Comparison of Mamdani-Type and Sugeno-Type Fuzzy Inference Systems for Air Conditioning System," *Int. J. Soft Comput. Eng.*, vol. 2, no. 2, 2012, pp. 323–325.
- [30] P. Cingolani and J. Alcalá-Fdez, "JFuzzyLogic: A robust and flexible Fuzzy-Logic inference system language implementation," in *IEEE International Conference on Fuzzy Systems*, 2012.
- [31] P. Cingolani and J. Alcalá-Fdez, "jFuzzyLogic: a Java Library to Design Fuzzy Logic Controllers According to the Standard for Fuzzy Control Programming," *Int. J. Comp. Int. Syst.*, vol. 6, no. sup1, 2013, pp. 61–75.
- [32] IEC, "Fuzzy Control Programming (IEC 1131-7 CD1)." pp. 1–53, 1997.

Fuzzy Bi-Objective Particle Swarm Optimization for Next Release Problem

Carlos Casanova*, Giovanni Daián Rottoli*, Esteban Schab*, Luciano Bracco*,
Fernando Pereyra* and Anabella De Battista*

* Computational Intelligence and Software Engineering Research Group (GIICIS)
Regional Faculty from Concepción del Uruguay.

National Technological University (UTN), Entre Ríos, Argentina.

Email: {casanovac, rottolig, schabe, braccol, pereyraf, debattistaa}@frcu.utn.edu.ar

Abstract—In search-based software engineering (SBSE), software engineers usually have to select one among many quasi-optimal solutions with different values for the objectives of interest for a particular problem domain. Because of this, a metaheuristic algorithm is needed to explore a larger extension of the Pareto optimal front to provide a bigger set of possible solutions. In this regard the Fuzzy Multi-Objective Particle Swarm Optimization (FMOPSO), a novel *a posteriori* algorithm, is proposed in this paper and compared with other state-of-the-art algorithms. The results show that FMOPSO is adequate for finding very detailed Pareto Fronts.

Index Terms—Search-Based Software Engineering; Multi-Objective Optimization; Particle Swarm Optimization; Next Release Problem; Fuzzy Logic.

I. INTRODUCTION

Search-Based Software Engineering (SBSE) is a discipline that aims to help software engineers build high quality software through the application of search methods. The main strategy is to change the focus from describing how to develop the software to describing what the software characteristics are. This description has to be codified to be understood by a search algorithm capable of generating new possible products and evaluate their quality using a set of rules provided by the engineer [1].

The problems to be solved using this type of approach are formulated as optimization problems that have, in the majority of the cases, a combinatorial search space and multiple objectives. Because of this, metaheuristics are generally used, discarding classical methods for optimization such as mathematical programming.

This paper introduces a first version of a novel metaheuristic algorithm named Fuzzy Multi-Objective Particle Swarm Optimization (FMOPSO), designed to deal with this kind of problems by creating a fitness function of multiple objectives using fuzzy weight factors. Different configurations of this fitness function are used to guide the method in the approximation of the Pareto-optimal front. This new algorithm has been tested on two instances of a well-known Search-Based Software Engineering problem, the Next Release Problem (NRP).

This problem, first proposed by [2], is aimed at finding a requirement subset to be implemented that satisfy the stakeholders' needs, looking for the maximization of the profit and minimization of the implementation cost [3]. In addition, it may also be restricted by dependencies between requirements such as precedence and simultaneity, among others.

The rest of this paper is organized as follows. In Section II the multi-objective optimization is introduced. Section III describes the Fuzzy Bi-Objective Particle Swarm Optimization algorithm proposed in this paper. Section IV explores the behavior of this proposal and compares it with another well known state-of-the-art algorithms. Finally, Section V contains the conclusions and future work.

II. MULTI-OBJECTIVE OPTIMIZATION

A commonly used optimization approach consists on selecting as objective function one of the system's attributes and using it to define the (total) order of preferences of the feasible solutions, resulting a *mono-objective* problem. The rest of the attributes modeled as constraints.

On the other hand, the *multi-objective* optimization approach uses several attributes as objective function. These objectives compete against each other defining a partial order on the solution space where there are solutions that are not comparable *a priori*. This partial order is called *Dominance Relation*. The set of all the non-dominated solutions is called *Pareto Front*, and is the result of an optimization method that makes no assumptions about the preferences of the decision-maker.

It is important for the Pareto Front to be as detailed as possible so the decision-maker can select the solution that best fits their needs. Additionally, the Pareto Front provides valuable information about the relation between the competing objectives to use to analyze "What if..." questions.

III. FUZZY MULTI-OBJECTIVE PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) is a population-based metaheuristic, which means that in each iteration there is a set (swarm) of possible solutions called particles that move through the search space to find new solutions.

The movement or change in the position of each particle is computed using a movement equation. This equation specifies the way in which the solution is perturbed. Thus, this is a perturbative metaheuristic.

Each particle has access to a fitness function that evaluates the efficiency of the particle's position. This function is the objective function to be optimized. Then, each particle i in the iteration $k+1$ change its position $X_i^{[k+1]}$ according to the particle's velocity $V_i^{[k+1]}$, computed using the movement rule:

$$V_i^{[k+1]} = V_i^{[k]} + w_C \times r_1^{[k+1]} \times [b_i^{[k]} - X_i^{[k]}] + w_S \times r_2^{[k+1]} \times [b_G^{[k]} - X_i^{[k]}] \quad (1)$$

where $b_i^{[k]}$ is the best position reached by the particle in previous iterations, $b_G^{[k]}$ is the best position reached by the swarm in previous iterations, r_1 and r_2 are random numbers uniformly distributed in the interval $[0,1]$, and w_C and w_S are prefixed constants. These three terms from the movement equation are, in that order, the inertia term, the memory term, and the cooperation term. Consequently, the position $X_i^{[k+1]}$ is modified as follow:

$$X_i^{[k+1]} = X_i^{[k]} + V_i^{[k+1]} \quad (2)$$

A. Canonical PSO Implementation

PSO is an algorithm that was originally designed to work with continuous variables. However, in order to approach combinatorial optimization problems, it can be reformulated to take into account the combinatorial aspects of the problem into the movement equation. It is because of this that the Canonical PSO model comes up [4], specifying the necessary and sufficient conditions to use PSO in any domain. According to it, the representation of the following items has to be defined: (a) the position and velocity of a particle; (b) a scalar-valued or vector-valued fitness function; (c) a total or partial order relation on the fitness function codomain; (d) the binary operations:

- $\text{subtraction}(\text{position}, \text{position}) \xrightarrow{-} \text{velocity}$
- $\text{external_product}(\text{real_number}, \text{velocity}) \xrightarrow{\cdot} \text{velocity}$
- $\text{addition}(\text{velocity}, \text{velocity}) \xrightarrow{+} \text{velocity}$
- $\text{displacement}(\text{position}, \text{velocity}) \xrightarrow{+} \text{position}$

For the Next Release Problem, binary vectors are used to represent the position and velocity of a particle [5]. Each vector has a length equal to the number of requirements. In the position vector, each component x_j represents the decision of including (1) or excluding (0) requirement j . In the velocity vector instead, the value 1 in the position v_j means the value in x_j has to change to its complement.

The binary operations are described in the following sections.

B. Fuzzy Single-Objectivization

It is possible to compose a single objective function of multiple objectives using fuzzy sets [6]. A fuzzy decision set \tilde{D} can be built from nO objectives $\tilde{O}_1, \tilde{O}_2, \dots, \tilde{O}_{nO}$ using the intersection according to the triangular norm t , that represents the objectives *confluence*. Thus, \tilde{D} is defined as follow:

$$\tilde{D} = \tilde{O}_1 \cap_t \tilde{O}_2 \cap_t \dots \cap_t \tilde{O}_{nO} \quad (3)$$

then, using an in-order notation:

$$\mu_{\tilde{D}}(x) = \mu_{\tilde{O}_1}(x) t \mu_{\tilde{O}_2}(x) t \dots t \mu_{\tilde{O}_{nO}}(x) \quad (4)$$

It is reasonable in the majority of the cases to choose the option x with the maximum membership degree to the fuzzy decision set.

Furthermore, [7] presents a mechanism to give to each objective differentiated preferences by affecting the membership function using an exponential weighting factor ρ in order to contract (increase), with $\rho > 1$, or dilate (decrease), with $\rho < 1$, the relevance of each objective.

Let $\rho_1, \rho_2, \dots, \rho_{nO}$ be the exponential weighting factors associated with each objective such that $\sum_{i=1}^{nO} \rho_i = nO$. In consequence, \tilde{D} is defined as follows:

$$\mu_{\tilde{D}}(x) = \mu_{\tilde{O}_1}^{\rho_1}(x) t \mu_{\tilde{O}_2}^{\rho_2}(x) t \dots t \mu_{\tilde{O}_{nO}}^{\rho_{nO}}(x) \quad (5)$$

The problem addressed in this paper contemplates two objectives: the implementation cost C , and the profit B given by the stakeholders' satisfaction. The membership function of the fuzzy number associated to the cost is:

$$\mu_C(X, \rho_C) = \begin{cases} 1 & \text{if } C(X) \leq C_{min} \\ \left(\frac{C(X) - C_{max}}{C_{min} - C_{max}} \right)^{\rho_C} & \text{if } C_{min} < C(X) < C_{max} \\ 0 & \text{if } C(X) \geq C_{max} \end{cases} \quad (6)$$

where $C(X)$ is the cost of implementing the solution X , C_{min} and C_{max} are the minimum and maximum values of reference for the cost variable, and ρ_C is the prefixed weighting factor.

In the same way, the membership function of the fuzzy number associated to the profit is:

$$\mu_B(X, \rho_B) = \begin{cases} 1 & \text{if } B(X) \geq B_{max} \\ \left(\frac{B(X) - B_{min}}{B_{max} - B_{min}} \right)^{\rho_B} & \text{if } B_{min} < B(X) < B_{max} \\ 0 & \text{if } B(X) \leq B_{min} \end{cases} \quad (7)$$

where $B(X)$ is the profit given by the solution X , B_{min} and B_{max} are the minimum and maximum values of reference for the profit variable, and ρ_B is the prefixed weighting factor.

If the stakeholders' preferences are not known and it is required to find the whole Pareto Front, this method is still suitable by using multiple weighting factor values to specialize the search in multiple regions of the search space at the same time.

This strategy may be implemented by assigning different values for the weighting factors to distinct groups of particles, so each group explores a different region of the space (Fig.

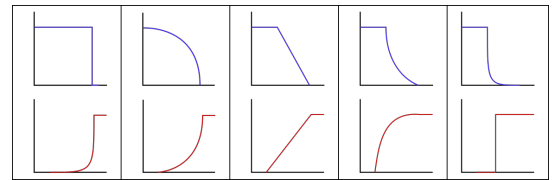


Fig. 1: Configurations of the fuzzy numbers for different groups (by column) with two objectives: Profit (blue): $\bar{\rho} = (0, 0.5, 1, 1.5, 2)$ and Cost (red): $\bar{\rho} = (2, 1.5, 1, 0.5, 0)$.

1). Therefore, the fitness function evaluated for a position X in group j is:

$$\mu_D(X, j) = \min \left(\mu_C \left(X, \frac{2(j-1)}{(n-1)} \right), \mu_B \left(X, 2 - \frac{2(j-1)}{(n-1)} \right) \right) \quad (8)$$

C. Topology and individual best updates

A key aspect to consider when using PSO is to define how the particles communicate with each other. The component that rules this communication is called *topology*.

Assuming that there are n different weighting factors values or, in other words, n groups into the swarm, the particles assigned to the j^{th} group share a single best group position denoted b_j that is used into the memory term of the movement equation. If each group has the same number of particles, say m , the swarm is thus arranged in a matrix $M^{m \times n}$. Consequently, each particle can be denoted with a double subindex (i, j) , with $1 \leq i \leq m$ and $1 \leq j \leq n$.

Additionally, each particle has access to the best position reached by its adjacent groups b_{j-1} and b_{j+1} , if they exist. In order to determine the position to be used in the cooperation term, the one with the highest value is selected.

Furthermore, an additional rule has to be taken into account to update the best group solutions b_j . When a new position $X_{ij}^{[k+1]}$ is computed, it is evaluated according to at most three version of the fitness function: $\mu_D(X_{ij}^{[k+1]}, j-1)$, $\mu_D(X_{ij}^{[k+1]}, j)$ and $\mu_D(X_{ij}^{[k+1]}, j+1)$.

As each particle is affected by its neighbours, this particle can find a good solution to its own fitness function version or to the version of its neighbors. Thus, when it is time to update the best group positions, b_j , the new position to be evaluated are $X_{il}^{[k+1]}$, with $i \in \{1, \dots, m\}$ and $l \in \{\max(1, j-1), \dots, \min(j+1, n)\}$.

D. Binary Operators

The binary operations used in the FMOPSO implementation are Xor, And & Or from Boole's Algebra, denoted by \oplus , \cdot , $+$, respectively. In this approach, the movement equation leave aside the inertia term:

$$V_{ij}^{[k+1]} = r_1^{[k+1]} \cdot (b_j^{[k]} \oplus X_{ij}^{[k]}) + r_2^{[k+1]} \cdot (b_{Vj}^{[k]} \oplus X_{ij}^{[k]}) \quad (9)$$

This rule can compute an unfeasible position, thus, the Xor operation related to the movement is performed in increasing order adding predecessors or removing successors, as appropriate.

Finally, if the particle does not change its position in two consecutive iterations, a mutation is applied on a random component to invert its value, adding predecessors or removing successors too.

IV. EXPERIMENTAL DESIGN

A study on two NRP instances obtained from the *classic* set of instances of [8] were performed. The first one, named *nrp1* has 140 requirements and 100 stakeholders. The second one, named *nrp2*, has 620 requirements and 500 stakeholders. Both instances have precedence relations between requirements. The reference values B_{min} and C_{min} were set to 0 for both

instances. The values for B_{max} were 2909 for *nrp1* and 14708 for *nrp2*. The values for C_{max} were 787 for *nrp1* and 4758 for *nrp2*. These values were found through an exact mono-objective optimization using Branch & Bound.

Several tests were conducted on the aforementioned instances using the proposed FMOPSO algorithm. The results were compared with two widely used state-of-the-art algorithms: NSGA-II (Non-Dominated Sorting Genetic Algorithm) [9] and IBEA (Indicator-Based Evolutionary Algorithm) [10].

The metrics used for the comparison were the Hypervolume (HV), to evaluate the quality of the solution, and the Pareto Front Size (PFS), to assess the population diversity [11].

The parameters were tuned ad-hoc by executing each algorithm 5 times with many different configurations to find the best set of values. The best configurations were selected according to the median of the Hypervolume. With this best setting for each algorithm, 10 more executions were performed to obtain a representative value distribution of the selected metrics. All the partial results can be found in [12].

The software used for the experiment was written in C++ using and extending the ParadisEO library [13]. FMOPSO was implemented by the authors. The ParadisEO's versions of NSGA-II and IBEA were used. The crossover operator used to produce two new individuals is equivalent to the boolean operators $+$ and \cdot . The mutation operator is the same used in the FMOPSO, as mentioned before. The code can be found in <https://github.com/casanovac/FMOPSO>.

The running time was the same for all the algorithms: 30 seconds for *nrp1* and 60 seconds for *nrp2*.

A. Results

The results (Table I) show that FMOPSO is a better option regarding diversity, this is, the values for the PFS metric, obtained using this algorithm, are higher than those obtained using the state-of-the-art alternatives. However, FMOPSO is overtaken by IBEA for the Hypervolume, but shows a better performance than NSGA-II, a widely used algorithm in Search-Based Software Engineering (Fig. 2).

Additionally, Figure 3 shows that NSGA-II does not cover the Pareto Front uniformly: there are many regions that were not explored. IBEA, on the other hand, generates the best non-dominated solutions, but with many gaps too. FMOPSO, however, scans the whole front in a uniform way, with a low

TABLE I: Result Descriptors

Metric	Problem Algorithm	FMOPSO	NRP1 IBEA	NSGAII	FMOPSO	NRP2 IBEA	NSGAII
HV	Min	0.5651	0.5586	0.5256	0.4550	0.4967	0.3952
	1 st Q.	0.5673	0.5614	0.5323	0.4570	0.4977	0.3996
	Median	0.5683	0.5652	0.5347	0.4592	0.4981	0.4032
	Mean	0.5679	0.5631	0.5351	0.4594	0.4982	0.4027
	3 rd Q.	0.5689	0.5646	0.5392	0.4612	0.4990	0.4052
	Max	0.5695	0.5668	0.5422	0.4663	0.4996	0.4106
PFS	Min	227	85	100	338	45	242
	1 st Q.	234	85	105.5	347	47.25	251.5
	Median	239	87.5	112	353.5	48	259
	Mean	240	88.22	110.2	353.2	47.9	257.8
	3 rd Q.	250	90.5	115	357.5	49	263
	Max	253	95	116	369	50	271

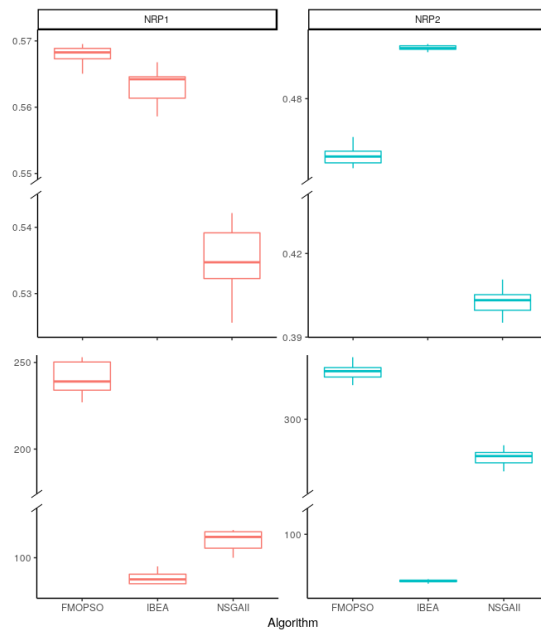


Fig. 2: Box plots with the distribution of the HV (Top) and the PFS (Bottom) on two instances of the Next Release Problem.

quality loss, as can be seen in the scatterplot related to the second instance of the problem.

According to this study, FMOPSO is an adequate alternative when the purpose is to obtain a very detailed Pareto Front, with a very good level of quality.

V. CONCLUSIONS AND FUTURE WORK

In this paper, the Bi-Objective Next Release Problem has been presented at a conceptual level. A novel *a posteriori* metaheuristic algorithm that approximates the Pareto Front for the aforementioned bi-objective NRP was also presented: FMOPSO. This algorithm composes a fitness function by using mixed weights for different groups of particles from the swarm.

Technical and theoretical aspects on this algorithm have been presented. A comparative study was performed comparing this proposal with two different state-of-the-art algorithms. The obtained results are promising and encourage to keep looking forward to the study of many yet unexplored aspects of this new algorithm.

It is necessary to adapt FMOPSO to be used in an interactive way. Extending the FMOPSO to deal with other SBSE problems that include more than two objectives should also be a goal.

ACKNOWLEDGEMENTS

Thanks to the National Technological University from Argentina (UTN) through the project SIUTICU0005297TC: "Preference-based Multi-objective Optimization Approaches applied to Software Engineering", and to Kevin-Mark Bozell Poudereux, for proofreading the translation.

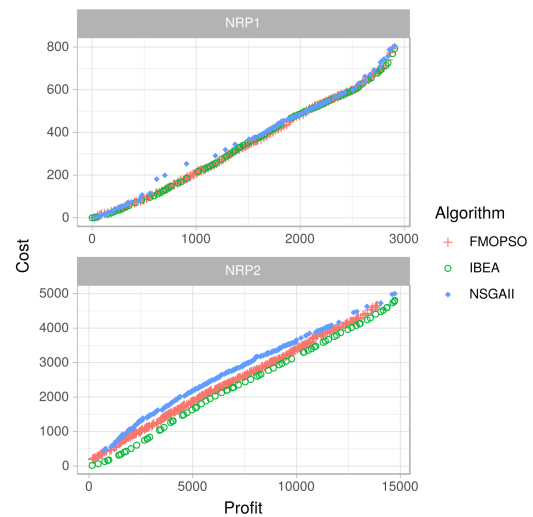


Fig. 3: Pareto Fronts from a random execution of the algorithms

REFERENCES

- [1] M. Harman and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.
- [2] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whitley, "The next release problem," *Information and Software Technology*, 2001.
- [3] Y. Zhang, M. Harman, and S. A. Mansouri, "The multi-objective next release problem," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07*. ACM Press, 2007, p. 1129.
- [4] M. Clerc, *Particle Swarm Optimization*, 2006.
- [5] F. Afshinmanesh, A. Marandi, and A. Rahimi-Kian, "A novel binary particle swarm optimization method using artificial immune system," in *Computer as a Tool, 2005. EUROCON 2005. The International Conference on*. IEEE, 2005, pp. 217–220.
- [6] R. E. Bellman and L. A. Zadeh, "Decision-making in a fuzzy environment," *Management science*, vol. 17, no. 4, pp. B–141, 1970.
- [7] R. R. Yager, "Multiple objective decision-making using fuzzy sets," *International Journal of Man-Machine Studies*, vol. 9, no. 4, pp. 375–382, 1977.
- [8] J. Xuan, H. Jiang, Z. Ren, and Z. Luo, "Solving the Large Scale Next Release Problem with a Backbone-Based Multilevel Algorithm," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1195–1212, sep 2012.
- [9] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *International conference on parallel problem solving from nature*. Springer, 2000, pp. 849–858.
- [10] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2004, pp. 832–842.
- [11] C. Grosan, M. Oltean, and D. Dumitrescu, "Performance metrics for multiobjective optimization evolutionary algorithms," in *Proceedings of Conference on Applied and Industrial Mathematics (CAIM), Oradea, 2003*.
- [12] C. Casanova, A. De Battista, E. Schab, G. D. Rottoli, L. Bracco, and F. Pereyra, "Execution Results Bi-Objective NRP with FMOPSO-IBEA-NSGAI," feb 2019. [Online]. Available: <https://dx.doi.org/10.17632/d7y3x97hsx.1>
- [13] S. Cahon, N. Melab, and E.-G. Talbi, "Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics," *Journal of heuristics*, vol. 10, no. 3, pp. 357–380, 2004.

Language Independent POS-tagging Using Automatically Generated Markov Chains.

Joaquim Assunção¹, Paulo Fernandes², Lucelene Lopes²

¹ UFSM Department of Applied Computing – Santa Maria – Brazil

² Roberts Wesleyan College – Rochester, NY – USA

joaquim@inf.ufsm.br, fernandes_paulo@roberts.edu, lopes_lucelene@roberts.edu

Abstract

This paper proposes a method to predict word grammatical classes using automatically generated discrete-time Markov chains to model typical sentences. Such method advantage relies on the availability of input resources needed to build an efficient and effective solution to virtually any language, dialect, or domain lingo. One of the main advantages of the proposed method is its simplicity when compared to other sophisticated approaches based on Hidden Markov Models or even more complex formalisms. The proposed method is instantiated to an example and we show that the achieved efficiency and effectiveness bring advantages to traditional similar solutions.

1 Introduction

Part-Of-Speech (POS) tagging is a very basic task for all Natural Language Processing (NLP) techniques based on linguistic approach. However, only well resourced languages have very effective solution for POS tagging [14]. This fact makes statistical approaches very popular to less resourced languages [6, 13] or even domain lingo [17]. Acquiring information on informal texts, as social networks chats and comments, the use of a tool independent of formal language definition is even more interesting, since it can help to tackle an open NLP problem.

In contrast with the difficulty to find out structured knowledge for less resourced and informal languages, the Internet offers abundant unstructured material (texts) in every language and dialect. Consequently, a procedure capable to produce structured knowledge from textual sources would cope with such limitations.

As for software tools, the basic modules needed for a POS-tagger are a dictionary retriever (to identify possible grammatical classes for known words) and a predictor (to disambiguate words that can be employed with more than one role, or guess the role for unknown words). The dictionary retrieving task for any language can be solved

very efficiently by the use of decision diagrams [12], and even multilingual dictionary retrievers can be implemented very efficiently [3].

The gap between a dictionary retriever and a POS-tagger is essentially the syntactic disambiguation task. In the context of this paper we limit ourselves to consider only a shallow POS-tagger assigning a word class to each word, but the methods and techniques discussed in this paper can be applied to even more sophisticated POS-tagger without any loss of generalization.

Our goal is to provide an effective grammatical class predictor to words inside a sentence previously tagged with an efficient dictionary retriever. In order to do so, we propose the construction of Markov chains [16] to describe typical phrases and use the transient analysis of such chains to predict the grammatical class of each word. These typical phrases can be viewed as a training set for the POS-tagger. Consequently, the produced Markov chains can be viewed as a model of the language patterns, *i.e.*, the structured knowledge of the target language. The enlargement of scope of POS-taggers would benefit several NLP tasks like term extraction [9, 10], ontology processing [5], and textual data mining [1], to cite a few.

Our proposed method fits into a POS-tagging landscape as simpler than sophisticated approaches based on Data mining approaches as Support Vector Machines - SVM [4], or inference models as Conditional Random Fields - CRF [8]. Nevertheless, our method is more flexible than traditional linguistic approaches heavily anchored in specific languages [2, 15].

2 Basic Tools

To better understand this paper proposed method, this section presents brief descriptions of Markov chains and Decision Diagrams-based dictionary retrievers.

2.1 Markov Chains Markov chains is a formalism to represent discrete state models as, in our case, a finite

—*DOI reference number: 10.18293/SEKE2019-097

automaton where the transitions are fired by the occurrence of a stochastic process [16]. Although there is a very large variety of Markovian models the plain discrete-time Markov chains are sufficient to express a model as a set of states and transitions among them associated to known probabilities. To exemplify such chains, as will be used in the context of this paper, Figure 1 depicts a simple chain with three states: Sunny, Cloudy, and Rainy.

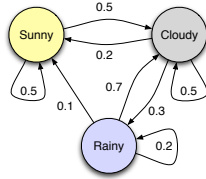


Figure 1: Simple Markov chain example.

Formally, we denote a DTMC by a matrix P where the element $p_{(i,j)}$ corresponds to the probability of leaving state i towards state j , e.g., in the chain of Fig. 1, $p_{(\text{Rainy}, \text{Cloudy})} = 0.7$ and $p_{(\text{Sunny}, \text{Rainy})} = 0.0$. For a model like this it is possible to make simple predictions, as for example, if a unknown day was preceded by a Sunny day and succeeded by a another Sunny day, it is possible to analyze all three possible intermediate states: Sunny-Sunny-Sunny ($0.5 \times 0.5 = 0.25$); Sunny-Cloudy-Sunny ($0.5 \times 0.3 = 0.15$); and Sunny-Rainy-Sunny ($0.0 \times 0.1 = 0.0$). Since all three possibilities sums up 0.4, it is possible to say that the intermediate day was: Sunny with probability 0.625 ($\frac{0.25}{0.4}$) or Cloudy with probability 0.375 ($\frac{0.15}{0.4}$).

2.2 Diagram Decision-based Dictionary Retriever The use of decision diagrams to recognize words, and associate its word class (or possible classes), can be very efficient and very effective as proposed in the WAGGER software tool [3]. According to this work, a decision diagram structure holding a multilingual dictionary with English and Portuguese words (a little more than one million words) can be used to tag possible word classes to large corpora (for instance, two million words) in less than three seconds using a personal machine. Such performance leads us to employ WAGGER for all experiments in our paper.

The technology behind such a dictionary retriever are Multi-Terminal Multi-valued Decision Diagrams [7]. Such structures are not only very effective to provide a fast recovery using small amounts of memory, but they also provide flexible structures that can be enhanced, for instance, by including new dictionary words.

The input of WAGGER is a dictionary in textual format, i.e., a list of words with their possible word classes. Then, it generates a MTMDD structure that can acts as dictionary retriever for a list of sentences that are annotated with the

possible word classes of each sentences' word.

I	believe	what	I	see	.	but
pronoun noun	verb	pronoun adverb	pronoun noun	verb	punctuation	conjunction preposition adverb noun
I	see	what	I	look	and	I
pronoun adverb	verb	pronoun adverb	pronoun noun	verb noun	conjunction noun	pronoun noun
look	what	I	want	to	look	.
verb noun	pronoun adverb	pronoun noun	verb noun	preposition adverb	verb noun	punctuation

Figure 2: WAGGER output for an example sentence.

The WAGGER output is all words of each sentence followed by the possible word classes. For example, Figure 2 presents the output of WAGGER using our English dictionary [3] for the sentence “I believe what I see, but I see what I look and I look what I want to look”. In this figure, the correct word class is indicated in bold. For instance, word “and” plays the role of conjunction in this sentence, but according to the dictionary “and” can also be employed as a noun.

In order to formalize the WAGGER output we will denote by w_i^s the i -th word in sentence s , and $C(w_i^s)$ the set of all possible word classes of word w_i^s . Note that the set $C(w_i^s)$ of a given word is independent of the sentence in which the word is (s). For instance, the word “and” (the thirteenth word of the example sentence) is formally defined by:

$$w_{13}^s = \text{“and”} \quad C(w_{13}^s) = \{\text{conjunction, noun}\}$$

3 Proposed Method

The proposed method follows the general idea of train and test, since it starts with a set of sentences manually tagged (training set), and only after we attempt to disambiguate word tags for sentences tagged by WAGGER (testing set). Figure 3 depicts the proposed method with the training task to construct the Markov chains, and the testing task to disambiguate WAGGER multiple or absent tags. It is important to notice that the training task needs a supervised action: a human made annotation, required according to the language of the target corpus.. All other tasks, including the annotation made by WAGGER, are fully automated, and, as exemplified in Section 4, memory and time efficient.

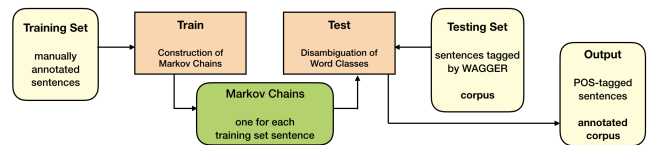


Figure 3: Proposed method.

3.1 Training Task Giving annotated sentences as the example in Figure 4, we construct a Markov chain for each sentence considering the sequences of word classes found in the sentence. This process corresponds to the capturing of an approximative model of the patterns of word class sequences, having each example sentence as an instance of the language usage.

Stochastic	models	will	save	the	world	.
adjective	noun	verb	verb	article	noun	punctuation
but	not	everything	in	our	imperfect	and
conjunction	adverb	pronoun	preposition	pronoun	adjective	conjunction
beautiful	world	deserves	to	be	saved	.
adjective	noun	verb	preposition	verb	verb	punctuation

Figure 4: Manually annotated sentence used as training.

Basically, the Markov chain created has all word classes as the chain states, and every exiting arc represents a possible succession of words classes. For instance, the adjectives (*adj*) are succeeded twice by a noun (*n*) and once by a conjunction (*conj*). Hence, the *adj* state has transition towards *n* with probability $\frac{2}{3}$ and a transition towards *conj* with probability $\frac{1}{3}$. Repeating the same procedure for all word classes creates the Markov chain depicted in Figure 5.

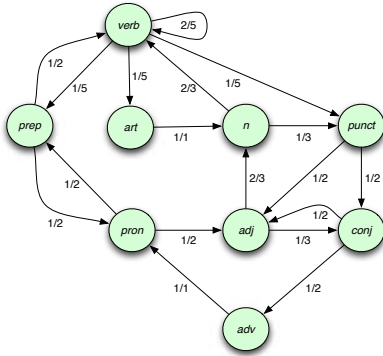


Figure 5: Markov chain for sentence in Figure 4.

This same process of creation of Markov chain is repeated for a certain number of sentences. For example, Figure 2 sentence correctly annotated would produce the Markov chain depicted in Figure 6.

3.2 Disambiguation using Markov Chains The disambiguation process is made through the estimation of probabilities of neighbors words according to the constructed chains. Every word that needs disambiguation is analyzed to all possible word classes, *i.e.*, we compute the probability of its immediate predecessor and successors according to each chain.

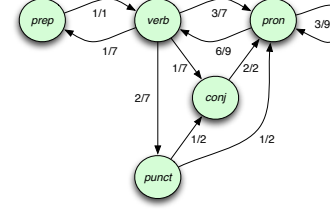


Figure 6: Markov chain for sentence in Figure 2.

Formally, a word w_i in sentence s , has its probability of being of word class c computed according its predecessor (Eq. 3.1) and successor (Eq. 3.2) to chain m respectively by:

$$(3.1) \quad \leftarrow c_{w_i^s}^{(m)} = \frac{\sum_{k \in C(w_{i-1}^s)} P_{[k(w_{i-1}^s), c(w_i^s)]}^{(m)}}{|C(w_{i-1}^s)|}$$

$$(3.2) \quad \rightarrow c_{w_i^s}^{(m)} = \frac{\sum_{k \in C(w_{i+1}^s)} P_{[c(w_i^s), k(w_{i+1}^s)]}^{(m)}}{|C(w_{i+1}^s)|}$$

where $P_{[x,y]}^{(m)}$ is the probability of leaving state x and going to state y in the Markov chain m .

The overall probability of a word w_i in sentence s be of class c according to all training chains of a set M is given by:

$$(3.3) \quad \hat{c}_{w_i^s} = \frac{\sum_{m \in M} \left(\leftarrow c_{w_i^s}^{(m)} + \rightarrow c_{w_i^s}^{(m)} \right)}{2 |M|}$$

Hence, the disambiguation of a word w_i^s will be made by choosing the word class $c \in C(w_i^s)$ with the maximum value $\hat{c}_{w_i^s}$.

For example, considering the sentence “*The old book is dusty and black.*”, the WAGGER annotation is depicted in Figure 7. In this sentence the words “*book*” - w_3^s (either a noun or a verb) and “*black*” - w_7^s (either an adjective, a noun or a verb) need disambiguation.

The	old	book	is
article	adjective	noun verb	verb
dusty	and	black	.
adjective	conjunction	adjective noun verb	punctuation

Figure 7: Example sentence to disambiguate.

Using the first chain (Figure 5), let us call it m_1 , we observe that the probability of a noun be preceded by an adjective (“*old*” precedes “*book*”) is equal to $\frac{2}{3}$, and since “*old*” can only be an adjective, *i.e.*, $C(\text{“old”}) = \{\text{adjective}\}$,

we express formally:

$$\overleftarrow{n}^{(m_1)}_{\text{"book"}} = \frac{P_{[adj,n]}^{(m_1)}}{|C(\text{"old"})|} = \frac{0.6667}{1} = 0.6667$$

We also observe that the probability of a noun to precede a verb ("book" precedes "is") is also equal to $\frac{2}{3}$, and since "is" can only be a verb, *i.e.*, $C(\text{"is"}) = \{\text{verb}\}$, we express formally:

$$\overrightarrow{n}^{(m_1)}_{\text{"book"}} = \frac{P_{[n,verb]}^{(m_1)}}{|C(\text{"is"})|} = \frac{0.6667}{1} = 0.6667$$

Analogously, repeating the computations for the second chain (Figure 6), let us call it m_2 , we obtain zero values, since there is no word class noun in it, formally:

$$\overleftarrow{n}^{(m_2)}_{\text{"book"}} = \frac{P_{[adj,n]}^{(m_2)}}{|C(\text{"old"})|} = \frac{0.0}{1} = 0.0$$

$$\overrightarrow{n}^{(m_2)}_{\text{"book"}} = \frac{P_{[n,verb]}^{(m_2)}}{|C(\text{"is"})|} = \frac{0.0}{1} = 0.0$$

Computing the overall probability for "book" be a noun considering the two Markov chains ($M = \{m_1, m_2\}$) will be formally:

$$\hat{n}_{\text{"book"}} = \frac{\overleftarrow{n}^{(m_1)}_{\text{"book"}} + \overrightarrow{n}^{(m_1)}_{\text{"book"}} + \overleftarrow{n}^{(m_2)}_{\text{"book"}} + \overrightarrow{n}^{(m_2)}_{\text{"book"}}}{2 |M|}$$

$$\hat{n}_{\text{"book"}} = 0.3333$$

Analogously, computing the probability of "book" being a verb in the sentence of Figure 7, we formally obtain:

$$\widehat{verb}_{\text{"book"}} = \frac{0.4}{4} = 0.1$$

As result, we conclude (correctly) that "book" must be tagged as a noun (probability 0.33), and not a verb (probability 0.1). Analogously, we conclude (also correctly) that "black" must be considered an adjective, since:

$$\widehat{adj}_{w_7^s} = 0.13 \quad \hat{n}_{w_7^s} = 0.08 \quad \widehat{verb}_{w_7^s} = 0.12$$

4 Application Example

We employ the proposed method in a larger scale to illustrate the efficiency and effectiveness benefits of our approach. To do so, we choose the following test bed:

- Two POS-tagger to Portuguese: PALAVRAS [2] and LX-Center Suite [15];
- A Portuguese dictionary and the WAGGER dictionary retriever [3];

- Fifty Brazilian Portuguese manually annotated sentences to be used as training set;

- One hundred and twenty eight Brazilian Portuguese manually annotated sentences to be used as test set.

Applying our proposed method to the sentences of the training set has produced 50 discrete-time Markov chains. As mentioned, these chains can be considered an approximative model of Brazilian language usage in terms of word class sequences. The choice of these sentences is not particularly planed, since these sentences were randomly taken from domain corpora composed of articles and other academic texts of several scientific domains [11].

After that, 128 sentences were also randomly chosen from the same corpora in order to test our proposed method. The only concern choosing these 128 test sentences were not to choose a sentence that was already used as training set. For both training and testing sets, the manual annotation was performed by two linguist specialists.

The application of our method to the training set delivered 2,455 correctly classified words from a total of 2,958 words from the testing set. This result corresponds to a 83.0% precision, which is comparable to the precision of word class tagging made by PALAVRAS (86.1%) and LX-Center (88.8%).

In terms of efficiency, however, our proposed method is far more impressive, while the time spent for PALAVRAS and LX-Center were measured in terms of minutes, the performance of our method took less than one second. Table 1 summarizes the effectiveness (precision) and efficiency (time to tag) of the prototype implementing our proposed method in comparison with PALAVRAS and LX-Center running on a portable machine with i7 2.2 GHz processor, 8 Gbytes memory. This table results indicate a reasonable effectiveness and an extraordinary efficiency improvement. Such performance let us believe that the proposed method is a worthy option for real-time POS-tagger.

Table 1: Overall comparative performance of the proposed method prototype.

	correct words	precision	time to tag
our method	2,455	83.0%	<1 sec.
PALAVRAS	2,548	86.1%	2.3 min.
LX-Center	2,627	88.8%	1.5 min.

Finally, it is important to remember that unlike PALAVRAS and LX-Center rigid approaches, our method is based on train and test phases. Therefore, a careful choice of a training set may improve the effectiveness of our method.

5 Final Considerations

This work lays down the basic ideas to a novel approach to predict grammatical classes from corpora. This novel idea is much simpler than sophisticated HMM, CRF or SVM. Nevertheless, our method is more prone to evolution than traditional rigid approaches available at existing POS-taggers like PALAVRAS and LX-Center Suite.

The examples presented here embody the core concept of our technique, yet the results delivered a reasonable precision and very impressive efficiency. We believe that a trusted Markov chain database can make a difference in this kind of scenario. Thus, we are currently working on the construction of a solid training databases for several formal and informal languages in order to promote a broader test. For instance, we are building an additional lexicon of informal words and abbreviations usually employed in informal chats to try to grasp a reasonable POS-tagging for such a complex dialect.

We also expect to refine the disambiguation phase by considering more information than just the predecessor and successor of the target word. It is important to call the reader attention that the number of Markov chains is not an obstacle to the efficiency, since the weight computed according to each Markov chain may be computed independently, *i.e.*, a parallel implementation of the proposed method would scale very well to a very large number of Markov chains.

Despite those promising future works, the initial results encourage this line of research. In fact, our approach seems to gather the flexibility of sophisticated learning methods, the easiness of development, and computational efficiency. Therefore, our proposed method can bring effective and efficient POS-tagging to virtually any language, dialect or domain lingo.

References

- [1] J. ASSUNÇÃO, P. FERNANDES, L. LOPES, AND S. NORMEY, *Distributed Stochastic Aware Random Forests - Efficient Data Mining for Big Data*, in 2013 IEEE Conference on Big Data, San Clara, CA, USA, June 2013, IEEE Computer Society, pp. 484–485.
- [2] E. BICK, *The parsing system PALAVRAS: automatic grammatical analysis of portuguese in constraint grammar framework*, PhD thesis, Arhus University, Arhus, Danemark, 2000.
- [3] P. FERNANDES, L. LOPES, C. A. PROLO, A. SALES, AND R. VIEIRA, *A fast, memory efficient, scalable and multilingual dictionary retriever*, in Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), N. Calzolari, ed., Istanbul, Turkey, may 2012, European Language Resources Association (ELRA), pp. 2520–2524.
- [4] J. GIMÉNEZ AND L. MÀRQUEZ, *Svmtool: A general pos tagger generator based on support vector machines.*, in LREC, European Language Resources Association, 2004.
- [5] R. GRANADA, L. LOPES, C. RAMISCH, C. TROJAHN, R. VIEIRA, AND A. VILLAVICENCIO, *A comparable corpus based on aligned multilingual ontologies*, in Proceedings of the First Workshop on Multilingual Modeling, 2012, pp. 25–31.
- [6] F. M. HASAN, N. UZ ZAMAN, AND M. KHAN, *Comparison of different pos tagging techniques (n-gram, hmm and brill's tagger) for bangla*, in Advances and Innovations in Systems, Computing Sciences and Software Engineering, K. Elleithy, ed., Springer Netherlands, 2007, pp. 121–126.
- [7] T. KAM, T. VILLA, R. K. BRYATON, AND A. SANGIOVANNI-VINCENTELLI, *Multi-valued decision diagrams: theory and applications*, Multiple-Valued Logic, 4 (1998), pp. 9–62.
- [8] J. D. LAFFERTY, A. MCCALLUM, AND F. C. N. PEREIRA, *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*, in Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01, San Francisco, CA, USA, 2001, Morgan Kaufmann Publishers Inc., pp. 282–289.
- [9] L. LOPES, P. FERNANDES, AND R. VIEIRA, *Estimating term domain relevance through term frequency, disjoint corpora frequency - tf-dcf*, Knowledge-Based Systems, 97 (2016), pp. 237 – 249.
- [10] ———, *Exato – high quality term extraction for portuguese and english*, in Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence, WI 2016, San Francisco, CA, USA, Oct. 2016, IEEE Computer Society, pp. 531–536.
- [11] L. LOPES AND R. VIEIRA, *Building domain specific parsed corpora in portuguese language*, in Proceedings of the X National Meeting on Artificial and Computational Intelligence (ENIAC), 2013, pp. 1–12.
- [12] C. L. LUCCHESI AND T. KOWALTOWSKI, *Applications of finite automata representing large vocabularies*, Software: Practice and Experience, 23 (1993), pp. 15–30.
- [13] C. D. MANNING, P. RAGHAVAN, AND H. SCHÜTZE, *Introduction to Information Retrieval*, Cambridge University Press, Cambridge, 2008.
- [14] F. SEGOND, A. SCHILLER, G. GREFFENSTETTE, AND J. CHANOD, *An experiment in semantic tagging using hidden markov model tagging*, in ACL/EACL Workshop on Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications, 1997, pp. 78–81.
- [15] J. SILVA, A. BRANCO, S. CASTRO, AND R. REIS, *Out-of-the-box robust parsing of portuguese*, in PROPOR 2010, 2010, pp. 75–85.
- [16] W. J. STEWART, *Probability, Markov Chains, Queues, and Simulation*, Princeton University Press, USA, 2009.
- [17] Y. TSURUOKA, Y. TATEISHI, J.-D. KIM, T. OHTA, J. MCNAUGHT, S. ANANIADOU, AND J. TSUJII, *Developing a robust part-of-speech tagger for biomedical text*, in Advances in Informatics, P. Bozanis and E. Houstis, eds., vol. 3746 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 382–392.

Generating SQL Statements from Natural Language Queries: A Multitask Learning Approach

Chunqi Chen, Yunxiang Xiong, Beijun Shen*, Yuting Chen

School of Electronic Information and Electrical Engineering

Shanghai Jiao Tong University, Shanghai, China

{15274521452, bruinx, bjshen, chenyt}@sjtu.edu.cn

Abstract—NL2SQL advocates an idea of helping engineers and/or end users generate SQL statements from natural language queries. However, it still remains a strong challenge in improving its precision and scalability. This paper introduces MultiSQL, a multitask deep learning approach to performing NL2SQL. MultiSQL unifies the task representations and trains a model in parallel on multiple tasks, including NL2SQL, machine translation, etc. It employs a multitask question-answering network for jointly learning all tasks and transferring knowledge among tasks. We have evaluated MultiSQL on two query datasets: WikiSQL (an open sourced dataset) and CnSQL (a Chinese dataset we created). The evaluation results clearly show the effectiveness of MultiSQL. In particular, the accuracies achieved by MultiSQL approximate those achieved by the state-of-the-art NL2SQL methods on WikiSQL, and its accuracy is 78%, which is 17% higher than the “Chinese2English + NL2SQL” method on CnSQL.

Index Terms—NL2SQL, SQL statement generation, deep learning, multitask learning

I. INTRODUCTION

With the rapid development of data engineering, industry engineers frequently perform data queries for data analyses and/or obtaining online reports. SQL is a popular and flexible language for querying data. In order to facilitate end users to perform data queries, some studies on NL2SQL, i.e., translating natural language queries into SQL statements, have been conducted.

So far two mainstreams of NL2SQL methods do exist. One mainstream is to use a *seq2seq* (sequence to sequence) model with neural networks for query generation [1], [2]. In particular, training a *seq2seq* model requires a standard format [2]. Meanwhile, a query result can have different logical forms, which reduces the effectiveness of the training process. *Seq2set* (sequence to set) is another mainstream, which divides a query into parts and generates them individually [3], [4]. A *seq2set* method is capable of processing the disorders of filtering conditions effectively, but it may miss internal dependencies in the input sequences.

The existing efforts have achieved remarkable results. However, they are still facing two main difficulties when applied in practice.

First, many NL2SQL methods are only evaluated on the machine-generated datasets, making their effectiveness unclearly shown. Indeed, most of the methods are evaluated on WikiSQL*—an open sourced dataset automatically extracted from Wikipedia. An NL2SQL method should be evaluated on both the machine- and the human-generated datasets [5].

Second, many NL2SQL methods are not scalable, as they only focus on translating queries in English into SQL statements. In case that queries in other languages are raised, they must at first be translated into descriptions in English, and then to SQL statements. The imprecision occurred during the translation stage can be propagated.

One solution to this is combining the language translation task with NL2SQL to seize the latent information in the queries and datasets. This paper presents MultiSQL, a multitask learning approach to NL2SQL. MultiSQL learns one model for multiple NLP tasks, including NL2SQL, machine comprehension, machine translation, etc. MultiSQL uses a TCR (Task-Content-Result) template to unify all tasks, and builds a multitask deep learning network for jointly learning all tasks and transferring knowledge among them.

This paper makes the next contributions:

- 1) **A deep learning approach.** We propose a general, scalable multitask deep learning approach, MultiSQL, to NL2SQL. MultiSQL trains a model for multiple NLP tasks, and takes three training strategies for accelerating sample efficient learnings and supporting knowledge transfers among tasks.
- 2) **A multitasking neural network.** We design a multitasking neural network with an encoder and a decoder. The encoder adopts dual coattention to represent the task and its content sequences, compressing all of this information with two BiLSTMs. The decoder with multi-pointer-generator utilizes attention over the content, task and previously output tokens to make decisions.
- 3) **Dataset and evaluation.** We have evaluated MultiSQL on WikiSQL and CnSQL. CnSQL is a Chinese SQL dataset we collected from real-world applications. Through transfer learning, the performance of NL2SQL is enhanced, achieving the logical form accuracy of 78.7% and the database execution accuracy of 86.1%. Furthermore, MultiSQL achieves logical form accuracy

*Corresponding author.

DOI reference number: 10.18293/SEKE2019-024

*<https://github.com/salesforce/WikiSQL>

of 78%, which is 17% higher than the “Chinese2English + NL2SQL” method.

II. RELATED WORK

A. Code Generation and NL2SQL

Code generation is becoming a promising approach to improve the productivity of software development. At present, there are three mainstream technologies: generation from models using templates [6], program synthesis by logic specification, and code generation and recommendation via machine learning or information retrieval [7]. NL2SQL is one of its research hotspots in recent decades, and in particular, neural-network-based NL2SQL has achieved remarkable results.

Dong and Lapata [1] have introduced a seq2seq approach that uses the augmented pointer network to convert textual queries to logical forms. Zhong et al. [2] have published the WikiSQL dataset and proposed a seq2seq model with reinforcement learning. Xu et al. [3] have further improved the results by taking a seq2set model and an attentional model. Similarly, Guo and Gao [8] have developed tailored modules to process three components in SQL queries. A parallel work [4] obtained a high execution accuracy on WikiSQL for SQL statements belongs to the “select-aggregator-where” type. External knowledge bases are also employed for tagging question words. Sun et al. [9] have presented a generative learning model to replicate contents in column names, cells, or SQL keywords. They have also improved the generation of the WHERE clauses by leveraging the column-cell relations.

B. Multitask Learning

Multitask learning is a machine learning paradigm. It aims to improve the generalization performance of a task using many other related tasks. Multitask learning has been successfully applied in the domain of natural language processing.

Collobert et al. [10] have proposed a unified framework for processing multiple natural language tasks, including chunking and part-of-speech tagging. Hashimoto et al. [11] have presented a neural network for dependency analysis, semantic correlation, and natural language reasoning. Zero-shot translation can be achieved by multitasking in language translations [12], where seq2seq models use two or more encoders and decoders for translation, parsing, and image subtitles [13]. Through model modularization, the above approaches can be applied to image classification and speech recognition [14]. Learning this modularity can further alleviate task interruptions [15]. Through multitask learning, the model can be used to learn some generally purposed expressions, since simultaneous learning of relevant tasks can provide inductive bias. Performance can be improved due to knowledge transfers across tasks.

III. APPROACH

MultiSQL is a multitask-learning-based approach to NL2SQL. As Fig. 1 shows, it takes a deep learning model to learn multiple tasks simultaneously. The deep learning model consists of a dual coattention encoder and a multi-pointer-generator decoder.

A. A Multitask QA Network

In MultiSQL, every task is formulated as a QA (Question Answering) task. Multiple tasks are trained jointly using a deep learning model, and knowledge is shared among tasks.

Task Unification. We design a TCR (Task-Content-Result) template to unify all the tasks. Each task instance is described with a *task*, *content*, and *result*, as Fig. 2 shows. During training, MultiSQL takes three sequences as its inputs: a content C with l tokens, a task Q with m tokens, and a result A with n tokens. Each token is represented by a d_{emb} -dimensional embedding.

Dual Coattention and Multi-Pointer. A task formed by the TCR template often contains key information that constrains the search space. MultiSQL uses dual coattention [16] for presenting conditions for both sequences, compresses the information with two BiLSTMs, applies self-attention [17] to collect long-distance dependencies, and then uses two BiLSTMs to get representations of the task and its content. The multi-pointer-generator [18] decoder uses attentions over the content, task, and previously output tokens to make a decision: copying from the content, copying from the task, or generating from a limited vocabulary.

Cross-task Transfer. MultiSQL facilitates sample-efficient learning and knowledge transfers among tasks. Three multitask collaborative training strategies can be used: *joint learning* (training all tasks jointly), *curriculum learning* (training simple tasks first), and *anti-curriculum learning* (training hard tasks first).

B. Encoder

MultiSQL adopts a deep stack-based recurrent neural network with a collaborative- and self-attention mechanism to generate the content embedding and the task embedding. The encoder takes six steps in the encoding process:

Step 1. Independent Encoding. A linear layer projects the input matrices onto a common d -dimensional space:

$$CW_1 = C_{proj} \in \mathbb{R}^{l \times d} \quad QW_1 = Q_{proj} \in \mathbb{R}^{m \times d}$$

The projected representations are fed into a shared BiLSTM $BILSTM_{ind}$ to get the independent encoded representations $C_{ind} \in \mathbb{R}^{l \times d}$ and $Q_{ind} \in \mathbb{R}^{m \times d}$.

Step 2. Alignment. The encoder obtains the coattended representations by aligning encoded representations of each sequence. The alignments are obtained by normalizing dot-product similarity scores between the content sequence and the task sequence:

$$\text{softmax}(C_{ind}Q_{ind}^\top) = S_{cq} \quad \text{softmax}(Q_{ind}C_{ind}^\top) = S_{qc}$$

Step 3. Dual Coattention. These alignments are used to compute weighted summations of a token in one sequence and a relevant token in another sequence:

$$S_{cq}^\top C_{ind} = C_{sum} \quad S_{qc}^\top Q_{ind} = Q_{sum}$$

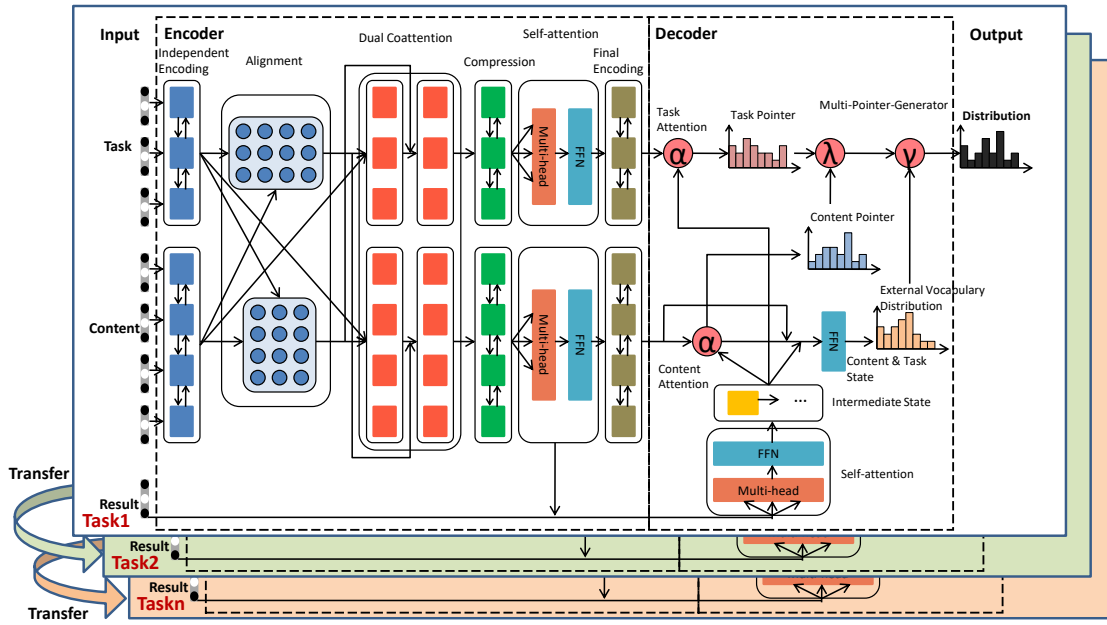


Fig. 1: An overview of the MultiSQL approach.

Task	Content	Result
Translate Chinese to English	告诉我南澳有哪些注意要点	Tell me what the notes are for South Australia
Generate SQL statements from neural text in English	The table has column names... Tell me what the notes are for South Australia	SELECT notes from table WHERE 'Current Slogan' = 'South Australia'

Fig. 2: Some TCR examples.

The coattended representations use the same weights to transfer information gained from alignments back to the original sequences:

$$S_{qc}^T C_{sum} = C_{coa} \quad S_{cq}^T Q_{sum} = Q_{coa}$$

Step 4. Compression. In order to compress embeddings from dual coattention back to the more manageable dimension d , we concatenate all of the four representations and feed them into separate BiLSTMs:

$$BiLSTM_{comC}([C_{proj}; C_{ind}; Q_{sum}; C_{coa}]) = C_{com}$$

$$BiLSTM_{comQ}([Q_{proj}; Q_{ind}; C_{sum}; Q_{coa}]) = Q_{com}$$

Step 5. Self-attention. We use multi-head, scaled dot-product attention [17] to capture long distance dependencies within each sequence.

$$Attention(\tilde{X}, \tilde{Y}, \tilde{Z}) = softmax\{\frac{\tilde{X}\tilde{Y}^T}{\sqrt{d}}\}\tilde{Z}$$

$$MultiHead(\tilde{X}, \tilde{Y}, \tilde{Z}) = [h_1; \dots; h_p]W^o,$$

where $h_j = Attention(\tilde{X}W_j^{\tilde{X}}, \tilde{Y}W_j^{\tilde{Y}}, \tilde{Z}W_j^{\tilde{Z}})$

All transformations in the above equations are linear such that the multi-head attention representations maintain dimensionality as d .

$$MultiHead_C(C_{com}, C_{com}, C_{com}) = C_{mha}$$

$$MultiHead_Q(Q_{com}, Q_{com}, Q_{com}) = Q_{mha}$$

We then use the projected, residual feedforward networks (FFNs). The FFNs are equipped with ReLU activations and layer normalization on the inputs and outputs (with parameters $U \in \mathbb{R}^{d \times f}$, $V \in \mathbb{R}^{f \times d}$):

$$FFN(X) = \max(0, XU)V + X$$

$$FFN_C(C_{com} + C_{mha}) = C_{self} \in \mathbb{R}^{l \times d}$$

$$FFN_Q(Q_{com} + Q_{mha}) = Q_{self} \in \mathbb{R}^{m \times d}$$

Step 6. Encoding. Finally, we feed C_{self} and Q_{self} into two BiLSTMs $BiLSTM_{finC}$ and $BiLSTM_{finQ}$ to get the representations $C_{fin} \in \mathbb{R}^{l \times d}$ and $Q_{fin} \in \mathbb{R}^{m \times d}$, respectively.

C. Decoder

MultiSQL's decoder takes four steps in decoding:

Step 1. Self-attention. The decoder starts by projecting the answer embeddings onto a d -dimensional space:

$$AW_2 = A_{proj} \in \mathbb{R}^{n \times d}$$

Since recurrence and convolution are not contained in this step, we add positional encodings to A_{proj} :

$$A_{proj} + PE = A_{ppr} \in \mathbb{R}^{n \times d},$$

$$\text{where } PE[t, k] = \begin{cases} \sin(\frac{t}{10000^{\frac{k}{2d}}}) & k \text{ is even;} \\ \cos(\frac{t}{10000^{\frac{k-1}{2d}}}) & \text{Otherwise.} \end{cases}$$

We use self-attention so that the decoder is aware of previous outputs and the content to prepare for the next output. A residual FFN layer is applied to the content:

$$\begin{aligned} MultiHead_A(A_{ppr}, A_{ppr}, A_{ppr}) &= A_{mha} \in \mathbb{R}^{n \times d} \\ MultiHead_{AC}((A_{mha} + A_{ppr}), C_{fin}, C_{fin}) &= A_{ac} \in \mathbb{R}^{n \times d} \\ FFN_A(A_{ac} + A_{mha} + A_{ppr}) &= A_{self} \in \mathbb{R}^{n \times d} \end{aligned}$$

Step 2. Getting Intermediate Decoder State. We next apply a standard LSTM with attention to get a recurrent content state \tilde{c}_t for time-step t . The LSTM produces an intermediate state h_t using the previous answer word A_{self}^{t-1} and recurrent content state:

$$LSTM([A_{self}^{t-1}; \tilde{c}_{t-1}], h_{t-1}) = h_t \in \mathbb{R}^d$$

Step 3. Task and Content Attention. This intermediate state is used to get attention weights α_t^C and α_t^Q , allowing the decoder to focus on encoded information of the time step t :

$$\begin{aligned} softmax(C_{fin}(W_2 h_t)) &= \alpha_t^C \in \mathbb{R}^l \\ softmax(Q_{fin}(W_3 h_t)) &= \alpha_t^Q \in \mathbb{R}^m \end{aligned}$$

Content representations are combined with these weights and fed through an FFN with \tanh activation to form the content state and the task state:

$$\begin{aligned} \tanh(W_4[C_{fin}^\top \alpha_t^C; h_t]) &= \tilde{c}_t \in \mathbb{R}^d \\ \tanh(W_5[Q_{fin}^\top \alpha_t^Q; h_t]) &= \tilde{q}_t \in \mathbb{R}^d \end{aligned}$$

Step 4. Multi-Pointer-Generator. The model may generate tokens that are not in the task or the content. Thus additional vocabulary tokens v are accessed. We obtain distributions over tokens in the task, content, and the external vocabulary:

$$\begin{aligned} \sum_{i: c_i = w_t} (\alpha_t^C)_i &= p_c(w_t) \in \mathbb{R}^n \\ \sum_{i: q_i = w_t} (\alpha_t^Q)_i &= p_q(w_t) \in \mathbb{R}^m \\ softmax(W_v \tilde{c}_t) &= p_v(w_t) \in \mathbb{R}^v \end{aligned}$$

Two scalar tune the importance of each distribution in determining the output distribution:

$$\begin{aligned} \sigma(W_{pv}[\tilde{c}_t; h_t; (A_{self})_{t-1}]) &= \gamma \in [0, 1] \\ \sigma(W_{cq}[\tilde{q}_t; h_t; (A_{self})_{t-1}]) &= \lambda \in [0, 1] \\ \gamma p_v(w_t) + (1 - \gamma)[\lambda p_c(w_t) + (1 - \lambda)p_q(w_t)] &= p(w_t) \end{aligned}$$

A token-level negative log-likelihood loss is used throughout the training process: $\mathcal{L} = -\sum_t \log p(a_t)$.

IV. EXPERIMENTS

We have implemented MultiSQL and evaluated it on several datasets. The evaluation is designed to answer the following research questions:

- **RQ1.** Is MultiSQL more effective than singletask learning methods?
- **RQ2.** Can MultiSQL be used to generate SQL statements from queries in other languages (Chinese in this study)?

A. Setup

The experiment contains ten NLP tasks, each of which is evaluated using one dataset. The ten datasets used in the evaluation are listed in Table I. In particular, NL2SQL was evaluated on WikiSQL (an open sourced dataset) and CnSQL (a Chinese dataset we created). CnSQL[†] includes 1534 pairs of Chinese queries and SQL statements. It was collected by the human engineers from end-user queries in a Chinese HR outsourcing platform (ezhiyang.com).

Several commonly used metrics are chosen: logical form accuracy [2] for NL2SQL, BLEU [19] for Chinese-English translation, and F1-Score for machine comprehension.

B. Results for RQ1

The pointer-generator seq2seq (S2S) model [18] is selected for comparison. The results for comparing the singletask and the multitask learnings are shown in Table I. Here, (w/SAtt), (+CAtt), (+QPtr), and (+ACurr) represent the S2S model with an adjunction of the self-attention mechanism, the dual co-attention mechanism, the task pointer, and the anti-curriculum learning strategy, respectively.

1) Singletask Learning

The self-attentive (w/SAtt) mechanism [17] increases the capacity of the S2S model of integrating information from the task and the content. It improves the performance on most of the tasks. For WikiSQL, this model approximates the state-of-the-art (72.4%), but it does not use a structured approach.

We supplement the model with a coattention mechanism (+CAtt) next. The performances on part of the tasks are improved while decrease on the others and significantly reduce on MNLI and MWSC. For these two tasks, since the S2S baselines have the content concatenated to the task, the pointer generator mechanism can copy words directly from the input. In case that the task and the content are separated, the effectiveness of the model may be reduced.

Thereafter, we add a task pointer (+QPtr) to the network, which boosts the performance on MNLI and MWS. It also improves performance on SQuAD to 75.5% in nF1, which matches the performance of the first wave of SQuAD models that utilize direct span supervision [16].

When tested on WikiSQL, the final model achieves logical form accuracy of 72.6% and database execution accuracy of 80.4%.

2) Multitask Learning

During multitask learning, the S2S model performs worse on many tasks than the singletask model, with a total score of 483.6 points. However, the performance of the model is getting better and better after the combination of (w/SAtt), (+CAtt), and (+QPtr) sequentially. The score finally reaches to 584.7 points.

Performances on tasks requiring heavy use of the external vocabulary drop more than 50% from the S2S baselines until (+QPtr) is added. In addition to a coattended content, the task pointer utilizes a coattended task, allowing task information

[†]<https://github.com/SimonCqChen/CnSQL>

TABLE I: Results for the singletask and multitask learnings.

Task	Dataset	Singletask				Multitask				
		S2S	w/SAtt	+CAtt	+QPtr	S2S	w/SAtt	+CAtt	+QPtr	+ACurr
NL2SQL	WikiSQL	60.0	72.4	72.3	72.6	45.8	64.8	72.9	74.0	78.7
Machine Translation	IWSLT	25.0	23.3	26.0	25.5	14.2	23.6	29.0	26.1	29.7
Machine Comprehension	SQuAD	48.2	68.2	74.6	75.5	47.5	66.8	71.8	70.8	74.3
Text Summarization	CNN/DM	19.0	20.0	25.1	24.0	25.7	14.0	15.7	23.9	24.6
Natural Language Inference	MNLI	67.5	68.5	34.7	72.8	60.9	69.0	70.4	70.5	69.2
Sentiment Classification	SST	86.4	86.8	86.2	88.1	85.9	84.7	86.5	86.2	86.4
Semantic Role Labeling	QA-SRL	63.5	67.8	74.8	75.2	68.7	75.1	76.1	75.8	77.6
Relationship Extraction	QA-ZRE	20.0	19.9	16.6	15.6	28.5	31.7	28.5	28.0	34.7
Goal Oriented Dialogue	WOZ	85.3	86.0	86.5	84.4	84.0	82.8	75.1	80.6	84.1
Semantic Parsing	MWSC	43.9	46.3	40.4	52.4	52.4	43.9	37.8	48.8	48.4
Total Score		-	-	-	-	483.6	566.4	543.8	584.7	607.7

to flow directly into the decoder. The main reason is that it is easy for the model to decide, if it can directly access the task, whether generating output tokens is more appropriate than copying.

By multitask joint learning with anti-curriculum strategy, the logical form accuracy of NL2SQL reaches 78.7%, and the database execution accuracy gets 86.1%.

C. Results for RQ2

The baseline for comparison is NL2SQL with Google’s translator. The results show that MultiSQL obtains logical form accuracy of 78% on CnSQL. It is 17% higher than that of the baseline. The results also indicate that MultiSQL performs well both on the human- and the machine-generated datasets.

V. CONCLUSION

MultiSQL is a multitask learning approach to generating SQL statements from natural language queries. MultiSQL leverages a deep learning model for jointly learning multiple tasks such that the performance of NL2SQL can get improved. The evaluation results show the effectiveness of MultiSQL. It achieves accuracies that approximate those of the state-of-the-art methods and outperforms the existing methods for translating Chinese queries into SQL statements.

In the future, we will improve MultiSQL by introducing other state-of-the-art language models. We also plan to feed domain knowledge into MultiSQL, expecting that the deep learning model can be further enhanced.

VI. ACKNOWLEDGEMENT

This research was sponsored by the National Key Research and Development Program of China (Project No. 2018YFB1003903), National Nature Science Foundation of China (Grant No. 61472242 and 61572312), and Shanghai Municipal Commission of Economy and Informatization (No. 201701052).

REFERENCES

- [1] L. Dong and M. Lapata, “Language to logical form with neural attention,” *arXiv preprint arXiv:1601.01280*, 2016.
- [2] V. Zhong, C. Xiong, and R. Socher, “Seq2sql: Generating structured queries from natural language using reinforcement learning,” *arXiv preprint arXiv:1709.00103*, 2017.
- [3] X. Xu, C. Liu, and D. Song, “Sqlnet: Generating structured queries from natural language without reinforcement learning,” *arXiv preprint arXiv:1711.04436*, 2017.
- [4] T. Yu, Z. Li, Z. Zhang, R. Zhang, and D. Radev, “Typesql: Knowledge-based type-aware neural text-to-sql generation,” *arXiv preprint arXiv:1804.09769*, 2018.
- [5] C. Finegan-Dollak, J. K. Kummerfeld, L. Zhang, K. Ramanathan, S. Sadasivam, R. Zhang, and D. Radev, “Improving text-to-sql evaluation methodology,” *arXiv preprint arXiv:1806.09029*, 2018.
- [6] F. Mao, X. Cai, B. Shen, Y. Xia, and B. Jin, “Operational pattern based code generation for management information system: An industrial case study,” in *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 425–430, IEEE, 2016.
- [7] S. Zhou, H. Zhong, and B. Shen, “Slampa: Recommending code snippets with statistical language model,” in *The 25th Asia-Pacific Software Engineering Conference (APSEC)*, 2018.
- [8] T. Guo and H. Gao, “Bidirectional attention for sql generation,” *arXiv preprint arXiv:1801.00076*, 2017.
- [9] Y. Sun, D. Tang, N. Duan, J. Ji, G. Cao, X. Feng, B. Qin, T. Liu, and M. Zhou, “Semantic parsing with syntax-and table-aware sql generation,” *arXiv preprint arXiv:1804.08338*, 2018.
- [10] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural language processing (almost) from scratch,” *Journal of machine learning research*, vol. 12, no. Aug, pp. 2493–2537, 2011.
- [11] K. Hashimoto, C. Xiong, Y. Tsuruoka, and R. Socher, “A joint many-task model: Growing a neural network for multiple nlp tasks,” *arXiv preprint arXiv:1611.01587*, 2016.
- [12] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, *et al.*, “Google’s multilingual neural machine translation system: Enabling zero-shot translation,” *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 339–351, 2017.
- [13] M.-T. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser, “Multi-task sequence to sequence learning,” *arXiv preprint arXiv:1511.06114*, 2015.
- [14] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, “One model to learn them all,” *arXiv preprint arXiv:1706.05137*, 2017.
- [15] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, “Learning what to share between loosely related tasks,” *arXiv preprint arXiv:1705.08142*, 2017.
- [16] C. Xiong, V. Zhong, and R. Socher, “Dynamic coattention networks for question answering,” *arXiv preprint arXiv:1611.01604*, 2016.
- [17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.
- [18] A. See, P. J. Liu, and C. D. Manning, “Get to the point: Summarization with pointer-generator networks,” *arXiv preprint arXiv:1704.04368*, 2017.
- [19] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318, Association for Computational Linguistics, 2002.

Forward Engineering Completeness for Software by Using Requirements Validation Framework

Nayyar Iqbal^{1,2,3}, Jun Sang^{1,2}, Min Gao^{1,2}, Haibo Hu^{1,2}, Hong Xiang^{1,2}

¹Key Laboratory of Dependable Service Computing in Cyber Physical Society of Ministry of Education, Chongqing University, Chongqing 400044, China

²School of Big Data & Software Engineering, Chongqing University, Chongqing 401331, China

³Department of Computer Science, University of Agriculture Faisalabad, Faisalabad 38000, Pakistan
nayyariqbal@cqu.edu.cn, jsang@cqu.edu.cn, gaomin@cqu.edu.cn, haibo.hu@cqu.edu.cn, xianghong@cqu.edu.cn

Abstract—In software development environment, software companies usually ignore the user requirements validation process in requirement gathering phase, which results in large number of modifications being required in the software maintenance phase to fulfill the customer requirements. Identification of accurate requirements from user stories and determining the effectiveness of work deliverable of software industry has always been a challenging task. In this paper, a new measurement approach for forward engineering completeness for software was introduced by using requirements validation framework. The forward engineering completeness for software was measured in two steps. In the first step, software component structure was developed in order to find the functional and non-functional requirements rejected by the customers in the requirement validation framework. In the second step, completeness of software from component-based development was determined in which the following parameters, such as functional, non-functional completeness attributes, were considered in the measurement process, and the unadopted attributes of the reuse code were also considered. Quality level for the attributes were assigned based upon the valuation of interior quality of the source code. Therefore, it resulted in the reduction of development time required for the software and the cost required for the software development was also reduced. A case study was incorporated in this research to explain the measurement process of forward engineering completeness. If the forward engineering code is satisfying the quality standards, then the code is in the completeness form. The attributes of code that negates to be used were considered as unadopted attributes.

Keywords—completeness; forward engineering; functional requirements; non-functional requirements; requirements engineering; validation

I. INTRODUCTION

It has been observed that software industries that abused the Requirement Engineering (RE) in the software development process resulted in the project failures [1-6]. Ana-Maria et al. [7] argued that business analysts needed to focus on requirement gathering techniques in a technically responsible way. Question had been raised as to the

relationship between the functional and non-functional requirements, with some, such as Vishal and Xiaoqing [8], argued that there was reciprocal relationship. Software quality measurement is one of the most complicated tasks in software design methodology [9]. Quality of the software can be determined by the success of software system for this various parameters, framework and methodologies has been proposed [10]. Forward Engineering as defined by Pressman [11] “In most cases, forward engineering does not simply create a modern equivalent of an older program. Rather, new user and technology requirements are integrated into the reengineering effort. The redeveloped program extends the capabilities of the older application”. Reverse and forward engineering are practiced in the legacy systems to extend the system usable lifespan [12].

Requirements are client’s invariant statements related to sub system or system [13]. Functional requirement describes the complete functionality of software components that should be required in the software. Non-functional requirement describes the requirements of software with respect to security, usability, portability, availability, capacity, efficiency and reliability [14]. In past lot of work has been done by the researchers on functional and non-functional requirements. But illustrating the graphical user interface for user requirements in the software specification, in order to validate the user requirements has always been ignored in the requirement gathering phase. This research focus on the importance of validation of user requirements so that time and budget wasted in the modification of software in the maintenance phase can be saved. Thalheim [15] suggested the design quality parameters which include completeness, naturalness, minimality and flexibility. The software after the development process is said to be in the completeness form, if it satisfies all the functional and non-functional requirements. Component is a reusable visible interface, which is the factored form of any software or sub system. Software architecture is a static structure that represents arrangement of components [13]. This research was conducted with the collaboration of software company. In this new template is introduced by the authors that demonstrates the requirements of software components, which is illustrated in Table 1. In

order to identify the adjusted and unadopted requirements Table 1 was discussed in the requirements validation framework.

In this research authors defines the two types of requirements unadopted and adjusted requirements. The functional or non-functional requirements rejected by Chief Executive Officer (CEO) of software users/customers are called unadopted requirements. Business analyst gathers the software requirements from software users/customers in natural language, after this these requirements were illustrated in Table 1, i.e. if software users/customers identifies that user login should not be by user name and password, but it must be by any of the followings: thumb scan, scan of Quick Response (QR) code or scanning the bar code of employee card etc. in the requirement validation framework then the rejected requirements are called unadopted requirements. Adjusted requirements are new requirements (functional or non-functional requirements) which are added in the software according to users demand or when any existing software components are replaced with new software components, then new functional and non-functional requirements are incorporated into the software. Examples includes: replacement of software component of login (email address and password) with QR code. In addition, as existing functionality of software was to calculate percentile of student result and now the customer of software has demanded that the software must also calculate Cumulative Grade Point Average (CGPA) of the student. Completeness is defined as “the state or condition of having all the necessary or appropriate parts” [16]. Whereas requirements completeness is defined as “a quality demanded to the set of software requirements and to each requirement itself, in order to ensure that there is no information left aside” [17].

In order to adapt the complete customer requirements in the software, software industries are developing the software globally [18]. The purpose of global software development is to gather the adjusted requirements of the software. As different countries use different social network software, such as Instagram, Twitter, Reddit, and Facebook etc., according to their requirements, therefore different regions in the world has different adjusted requirements. These differences are due to cultural difference, language difference, platform difference, business process difference and how the organization interpret with manual work. Different countries have different cultural and business-related problems so there is need to develop the software that captures the complete organization processing tasks. For this adjusted requirement must be incorporated in the requirement gathering phase, so that the developed software must be in the complete form.

II. METHODOLOGY

In this research authors develops the software by using forward engineering completeness approach. The methodology structure is illustrated in the Fig. 1. The developed system is said to be in forward engineering completeness, if it is developed with complete conditions or states of new business procedures and rules according to software engineering philosophies. In System Specification (SyS) information related to functional requirements, data

requirements, quality requirements and constraints for software was determined. Problem definition, objectives, goals, context and major capabilities of the software was determined.

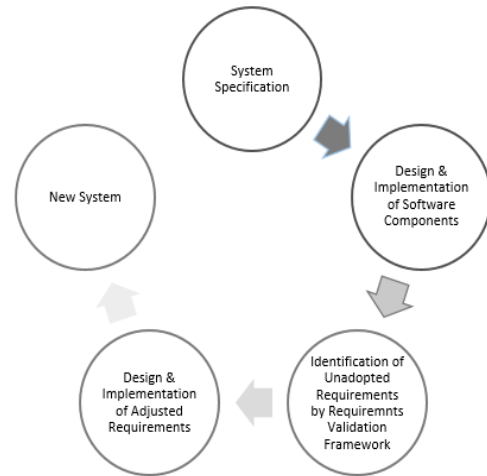


Figure 1. Forward Engineering Completeness

The purpose of requirement validation framework was to identify unadopted requirements as illustrated in the Fig. 2. In this business requirements were gathered from the users in which user identifies the goal and objectives of the system.



Figure 1. Requirements Validation Framework

The business analyst specifies the functional and non-functional requirements of the system. The business analyst and software engineer completed the task of software specification as shown in Table 1. The Software Quality Assurance (SQA) team members performed testing on the software to identify the errors in the software components. The basic purpose of this Table 1 was to present it in the framework meeting so the different categories of end-users from different regions can present their views. Scribe writes the report of the meeting. Chief executive officer attended the meeting along with end-users. Project manager and development team leader monitored the complete working process from requirement gathering to validation process. Software developer delivers the presentation of software requirements. The advantage of using this requirement validation framework showed the successful completion of software because after this process software modifications were not required in the software maintenance phase.

Rules & Definitions

Where S stands for Software, R_i , FR_j , NFR_k stands for n number of Requirements, Functional Requirements and Non-Functional Requirements respectively. Where $1 \leq i \leq n$, $1 \leq j \leq n$ and $1 \leq k \leq n$.

$$S(R_1, R_2, R_3, \dots R_n)$$

whereas

$$FR_j \& NFR_k \in R_i$$

As defined by Sommerville [19]

S ("what a software should do" & "how the system will do so")

Therefore

$$S(FR_1 \& NFR_1, FR_2 \& NFR_2, FR_3 \& NFR_3, \dots FR_n \& NFR_n)$$

Unadopted and adjusted requirements can be functional or non-functional requirements. Unadopted functional, unadopted non-functional, adjusted functional and adjusted non-functional requirements are represented by UFR_i , $UNFR_m$, AFR_p and $ANFR_q$ respectively. $SFEC$ stands for software developed by forward engineering completeness approach. Where $1 \leq l < n$, $1 \leq m < n$, $1 \leq p < n$ and $1 \leq q < n$.

$$FR_j \& NFR_k \in S \text{ and}$$

also

$$UFR_l \& UNFR_m \in S$$

but

$$AFR_p \& ANFR_q \notin S$$

whereas

$$AFR_p \& ANFR_q \in SFEC$$

In order to measure the Forward Engineering Completeness for the software this research incorporates case study and two steps were considered in the measurement process. In the first step, unadopted requirements were identified in the requirement validation framework and for adjusted requirements, structure of software component was also discussed. In the second step, software completeness was calculated by using following parameters. *i-First parameter* was functional and non-functional requirements attributes. These attributes determine the completeness of the forward engineering. In this integration among the attributes was also determined, which increases the completeness value. More completeness scales the forward engineering process closer to the actual budget and development schedule of the software and vice versa. *ii-Second parameter* was unadopted attributes. In which authors identified those attributes that were not required in the software by requirement validation framework. The identification of unadopted attributes helps in budget and time saving. The time spent on developing the software components that were not required in the developed software, was saved by identifying the unadopted attribute in the initial phase.









As clarified by Sommerville [19] "the non-functional requirements should define the usability, security, availability and performance requirements of the service". Therefore, usability and security were important for each component in non-functional requirements. The introduced technique was helpful for software engineers to measure the forward engineering completeness for software. In forward






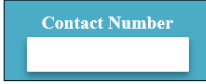


engineering, the software changes can be in terms of technology or adding new functionality e.g. if software was developed in old technology it can be changed to new technology for this, functional, non-functional and unadopted requirements were identified according to plate form difference.

III. CASE STUDY

Table 1 consists of three columns, first column describes the functional requirements of software, second column describes the system response and the third column illustrates the software components structure. Table 1 was discussed in the requirement validation framework. The non-functional requirements where were applicable, were explained by using software component structures in the requirement validation framework. In the pre-condition, user login the system by using organization email address. The organization email addresses and default password were issued by the organization for their employees.

TABLE I. REQUIREMENTS OF SOFTWARE COMPONENTS

Functional Requirements	System Response	Software Components
FR₁: The system shall display text box to enter the email address	The system displays a message asking the user to enter organization email address	
FR₂: The system shall display text box to enter the password	The system displays a message asking the user to enter the password	
	The system displays a message of "Successful Login"	
	The system displays the user Employee Number (EN) and name after the login	
FR₃: The system shall display options (yes/no) to change the password	The system asks the user if he/she wants to change the password	
	If the user clicks on the yes option, the system displays a message asking the user to enter current password, new password and confirm new password	
	The system displays the message "Password Changed Please login again with new Password"	
FR₄: The system shall display drop-down list for the selection of department	The system displays a message asking the user to select the department from the drop-down list	

FR₅: The system shall display drop-down list for the selection of employee title	The system displays a message asking the user to select employee title from the drop-down list	
FR₆: The system shall display calendar control for the selection of date of joining	The system displays a message asking the user to select date of joining the organization from the calendar control	
FR₇: The system shall display drop-down list for the selection of bank title	The system displays a message asking the user to select bank title from the drop-down list	
FR₈: The system shall display text box to enter bank account number	The system displays a message asking the user to enter bank account number	
FR₉: The system shall display text box to enter home address	The system displays a message asking the user to enter home address	
FR₁₀: The system shall display text box to enter contact number	The system displays a message asking the user to enter contact number	
FR₁₁: The system shall display options (yes/no) to save the changes	The system displays a message asking the user whether he/she wants to save the required data in the software or not	
	The system displays the message "Changes Saved" if user clicks on the yes option	

A. Condition 1

Customer₁ from organization₁, requested the modification in the software, functional and non-functional requirements for user login were by scanning the bar code of employee card with the bar code reader instead of login by email address and password. The bar code reader will be connected with the system through serial port or interface device called wedge or keyboard port. The bar code of the card will be matched with the repository of the user saved in the software in order to find the user matching text (identification).

According to Table 1, Functional Requirement Attributes (*FRA*) are those that defines the system behavior under precise circumstances. *FRA* (*email_address*, *password*, *change_password*, *current_password*, *new_password*, *confirm_new_password*, *department*, *employee_title*, *date_of_joining*, *bank_title*, *bank_account_number*, *home_address*, *contact_number*, *save_changes*). Non-functional Requirement Attributes (*NFRA*) are those that defines in what way a system must act and create restraints on its functionality. *NFRA* (*security*, *usability*, *portability*, *availability*, *capacity*, *efficiency*, *reliability*, *performance*, *integrity*, *recovery*, *compatibility*, *maintainability*). Unadopted Attributes, Total functional and non-functional requirements Attributes are represented by *UA* and *TA* respectively. In this following were the unadopted attributes *email_address*, *password*, *change_password*,

current_password, *new_password*, *confirm_new_password*. *FRA* = 14, *NFRA* = 12, *UA* = 6, *TA* = 26.

Functional Requirement Attributes Completeness

$$(FRAC) = FRA/TA = 14/26 = 0.54$$

Non-Functional Requirements Attributes Completeness

$$(NFRAC) = NFRA/TA = 12/26 = 0.46$$

Unadopted Attributes Completeness

$$(UAC) = UA/TA = 6/26 = 0.23$$

Software Completeness = *FRAC* + *NFRAC* - *UAC*

$$= 0.54 + 0.46 - 0.23 = 0.77$$

As the value are represented in unit interval (0, 1).

B. Condition 2

Customer₂ from organization₂, whose functional and non-functional requirement were: when the user login the system by email address and password. The system shall send PIN at the user cellphone for further authentication of user. So, there was requirement of new functionality by the user to be added in the software. The new functionality was required to be integrated with email software component. According to Table 1, *FRA* = 14, *NFRA* = 12, *UA* = 1, *TA* = 26.

Functional Requirement Attributes Completeness

$$(FRAC) = FRA/TA = 14/26 = 0.54$$

Non-Functional Requirements Attributes Completeness

$$(NFRAC) = NFRA/TA = 12/26 = 0.46$$

Unadopted Attributes Completeness

$$(UAC) = UA/TA = 1/26 = 0.04$$

Software Completeness = *FRAC* + *NFRAC* - *UAC*

$$= 0.54 + 0.46 - 0.04 = 0.96$$

C. Condition 3

Customer₃ from organization₃, functional and non-functional requirements for user login were by thumb scan or by scanning the Quick Response (QR) code instead of login by email address and password. QR code functionalities are represented in the Fig. 3 [10].

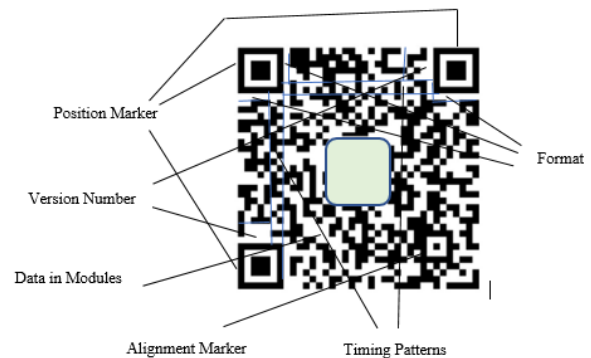


Figure 3. QR Code

According to Table 1, the value of *FRA*, *NFRA*, *UA* and *TA* were same as for condition 1, because unadopted attributes in both conditions were same.

Software Completeness = *FRAC* + *NFRAC* - *UAC*

$$= 0.54 + 0.46 - 0.23 = 0.77$$

D. Condition 4

Customer₄ from organization₄, functional and non-functional requirements were by selecting images for the password. The images must be available in the software. According to Table 1, $FRA = 14$, $NFRA = 12$, $UA = 4$, $TA = 26$. In this unadopted attribute were *password*, *current_password*, *new_password*, *confirm_new_password*.

Functional Requirement Attributes Completeness

$$(FRAC) = FRA/TA = 14/26 = 0.54$$

Non-Functional Requirements Attributes Completeness

$$(NFRAC) = NFRA/TA = 12/26 = 0.46$$

Unadopted Attributes Completeness

$$(UAC) = UA/TA = 4/26 = 0.15$$

$$\text{Software Completeness} = FRAC + NFRAC - UAC \\ = 0.54 + 0.46 - 0.15 = 0.85$$



Figure 4. Images Displayed for Password

In this user selects six images for password according to his/her order.



Figure 5. Images Selected for Password

In this research, complete software was developed in which team A used Forward Engineering (FE) approach. Team B developed the software by using Forward Engineering Completeness (FEC) approach. Both approaches were evaluated by monitoring following types of errors/defects [11]: Incomplete or Erroneous Specification (IES), Misinterpretation of Customer Communication (MCC), Intentional Deviation from Specification (IDS), Inconsistent Component Interface (ICI), Miscellaneous (MIS). Total number of Correct Functionalities (CF) in the software were also counted. In this research only IES, MCC, IDS, ICI, MIS, were monitored, because these errors/defects were related to the introduced technique. Total number of functional and non-functional requirements in the software were 1398. Table 1 represents the basic software components. During the evaluation process following errors in the software were identified: $IES = 173$, $MCC = 122$, $IDS = 27$, $ICI = 42$, $MIS = 301$, $CF = 733$ as shown in Fig. 6. All other errors/defects that does not belong to IES, MCC, IDS, ICI, were considered in the MIS. One value was assigned for one error/defect whether that error/defect belongs to functional or non-functional requirements of 1398 total requirements.

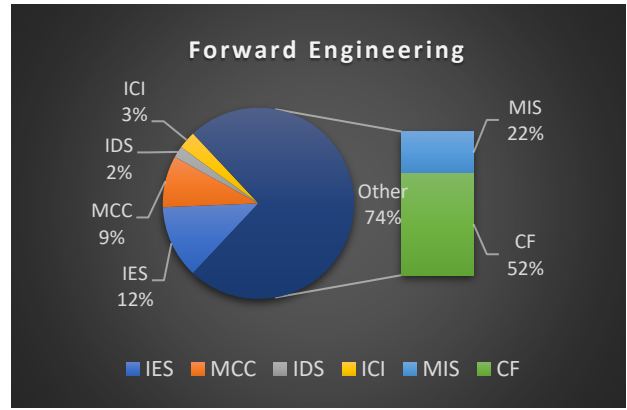


Figure 6. Errors & Correct Functionalities in FE Approach

In forward engineering completeness same type of errors/defects were monitored in the development process in order to determine the importance of requirement validation framework. Total number of functional and non-functional requirements were in the range of 1398 to 1430. The range in requirements were due to modifications in the unadopted requirements in order to fulfill different customer needs. Maximum value of requirements was assigned to the total requirements. During the evaluation process total number of errors in the software were as: $IES = 11$, $MCC = 14$, $IDS = 9$, $ICI = 21$, $MIS = 39$, $CF = 1336$ as shown in Fig. 7.

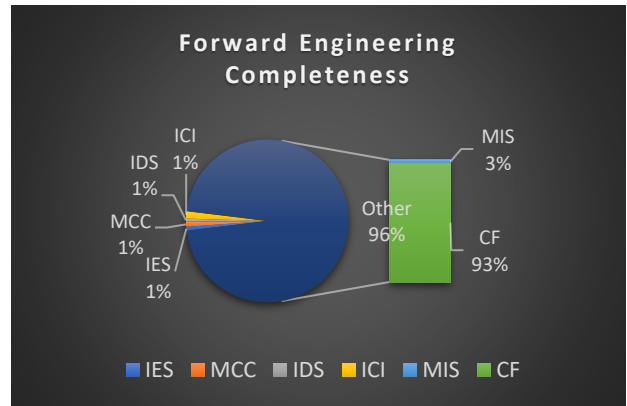


Figure 7. Errors & Correct Functionalities in FEC Approach

Team A developed the software without following the introduced techniques whereas team B followed the template of table for software requirements. After this these requirements were validated in the requirement validation framework before the actual development of software. Team A software development duration was more than the prescribed duration whereas team B developed the software in less than the prescribed duration. Total percentage of errors in software requirements was about 48% in forward engineering approach. In forward engineering completeness approach total percentage of errors in software requirements was about 7%. The decrease in errors was due to the validation of requirements before the software development. It has been observed that if development time of software increase, budget allocated for that software becomes less. As team A completed the software two months more than prescribed duration, so for these two months extra budget was

used in order to fulfill the salaries requirements of employees and other expenses of software company. Software development time increases due to identification of errors and defects in software, if these are found in last phases of system development life cycle than more time is required to remove them. As team B developed the software by using requirement validation framework therefore the defects found in this were nearly negligible. From the Fig. 7 it has been observed that whenever unadopted requirements are identified at the start of software development, there was reduction in budget and time duration for development also reduces. Forty-five days were required by Team A to perform corrective, adaptive, perfective and preventive maintenance whereas Team B completed all maintenance types in one day.

IV. CONCLUSIONS

This research supported the convincing evidence that, whenever requirement validation framework was used in the forward engineering, then surplus budget allocated for the maintenance phase was saved. The suitability of attributes allows illustrating conclusion about how suitable software component was for a specific problem. The time required to develop the software was also reduced. The time spends on corrective, adaptive, perfective and preventive maintenance reduces approximately 1 to 2 months for one-year projects, whereas in normal routine it takes 2 to 3 times more than scheduled time. It has been observed that software size is increasing day by day due to change in technology and new requirements of end-users. As software size increases ultimately the software complexity also increases. In the final phase the software, size becomes like a pyramid so if user stories are ignored in the requirement gathering phase then large number of errors and defects are identified in the software. The requirement validation framework identified the unadopted requirement in the software and new requirement were also identified in the face to face meeting which resulted the software in completeness form. The identification of unadopted requirement saved the software engineers from complexity of errors and defects.

ACKNOWLEDGMENT

This work was supported by Chongqing Research Program of Basic Science & Frontier Technology with Grant No. cstc2017jcyjB0305.

REFERENCES

- [1] S. Lauesen, "Problem-Oriented Requirements in Practice -A Case Study", In REFSQ 2018, Utrecht, Netherlands, 2018, pp. 3-19.
- [2] R. B. Svensson, T. Gorschek, B. Regnell, R. Torkar, A. Shahrokni, R. Feldt, "Quality requirements in industrial practice - an extended interview study at eleven companies," *IEEE Trans. Softw. Eng.* Vol. 38, 2012, pp. 923-935.
- [3] R. Berntsson-Svensson, T. Olsson, B. Regnell, "An Investigation of How Quality Requirements are Specified in Industrial Practice", *Inf. Softw. Technol.* vol. 55, 2013, pp. 1224-1236.
- [4] K. Wnuk, R. K. Kollu, "A Systematic Mapping Study on Requirements Scoping", In: *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, Limerick, Ireland, 2016.
- [5] P. Zave, M. Jackson, "Four Dark Corners of Requirements Engineering", *ACM Trans. Softw. Eng. Methodol.* 6(1), 1997, pp. 1-30.
- [6] S. Hotomski, E. B. Charrada, M. Glinz, "An Exploratory Study on Handling Requirements and Acceptance Test Documentation in Industry", In: *24th IEEE International Requirements Engineering Conference*, Beijing, China, 2016, pp. 116-129.
- [7] G. Ana-Maria, O. Cristina-Claudia, A. B. Robert, "Security Requirements Elicitation from Engineering Governance, Risk Management and Compliance", In *REFSQ 2018*, Utrecht, Netherlands, 2018, pp. 283-289.
- [8] S. Vishal, F. L. Xiaoqing, "Analysis of Conflicts Among Non-Functional Requirements Using Integrated Analysis of Functional and Non-Functional Requirements", In *31st Annual International Computer Software and Applications Conference*, IEEE Computer Society, Beijing, China, 2007, pp. 215-218.
- [9] M. K. Chawla, I. Chhabra, "A Quantitative Framework for Integrated Software Quality Measurement in Multiversions Systems," *International Conference on Internet of Things and Applications*, IEEE, Pune, India, 2016, pp. 310-315.
- [10] M. Yan, X. Xia, X. Zhang, L. Xu, D. Yang, "A Systematic Mapping Study of Quality Assessment Models for Software Products," *International Conference on Software Analysis, Testing and Evolution*, IEEE, Harbin, China, 2017, pp. 67-71.
- [11] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill Education, New York, USA, 8th ed, 2014.
- [12] M. Rouse, "Legacy Platform", Internet: <https://whatis.techtarget.com/definition/legacy-platform-legacy-operating-system>, March 2011, [February 22, 2019].
- [13] D. Gustafson, *Schaum's Outlines Software Engineering*, McGraw-Hill Education, New York, USA, 1st ed, 2002.
- [14] X. Lian, L. Zhang, "Optimized Feature Selection Towards Functional and Non-Functional Requirements in Software Product Lines", *IEEE, 22nd International Conference on Software Analysis, Evolution, and Reengineering*, Montreal, QC, Canada, 2015, pp. 191-200.
- [15] B. Thalheim, *Entity-Relationship Modeling: Foundations of Database Technology*, Springer-Verlag, Berlin, Germany, 1st ed, 2000.
- [16] G. D. S. Hadad, C. S. Litvak, J. H. Doorn, M. Ridao, *Dealing with Completeness in Requirements Engineering*, McGraw-Hill Education, New York, USA, 5th ed, 2015.
- [17] M. K. Pour, "Encyclopedia of Information Science and Technology", IGI Global, USA, 4th ed, 2017.
- [18] J. Noll, S. Beecham, I. Richardson, *Global Software Development and Collaboration: Barriers and Solutions*, *ACM Inroads - Special Section on Global Intercultural Collaboration*, 1(3), 2010, pp. 66-78.
- [19] I. Sommerville, *Software Engineering*, Pearson Education Limited, London, England, 10th ed, 2015.

Improving Mobile Device Interaction for Parkinson's Disease Patients via PD-Helper

Farzana Jabeen^{1,2}, Linmi Tao¹, Yirou Guo³, Shiyu Zhang⁴, Shanshan Mei⁵

¹Key Laboratory of Pervasive Computing, Ministry of Education, Beijing China

¹Dept. of Computer Science & Technology, Tsinghua University, Beijing China

²National University of Science and Technology, Islamabad, Pakistan

³Department of Electronic Engineering, Beihang University; ⁴International School, BUPT, Beijing, China

⁵Neurology Department, Xuan Wu Hospital, Beijing, China

Abstract—In current era, smart devices play vital role in social connectivity and performing routinely activities. The motor and non-motor (rigidity, tremor at rest, dysarthria, slowness of movement, and depression) symptoms of Parkinson's disease (PD) have definite negative impacts on patients' interaction with smart devices. An adaptable interface named PD-Helper is designed and developed to meet the physical capabilities as well as social needs of patients and to improve the accessibility via smart devices. Single touch interaction is provided via scanning selection technique, video and web user controls and input method editor (IME) application allows PD patients to access web browsing, entertainment like watching videos, typing text and contacting medical stuff. 40 users with PD, participated in the evaluation process of the PD-Helper. Preliminary results are promising and satisfactory. Further studies are in progress to attest the PD helper potential, such as improving the quality of life of people with other Neurological disorders.

Keywords—component; Parkinson's Disease, Mobile Interaction, Accessibility, Single touch interaction, interaction with smart devices

I. INTRODUCTION

In current society, Parkinson's disease (PD) is a progressive degenerative disorder that affects the nervous system and consequently decreases the quality of life (QOL) effecting more than 1% of the patients worldwide and in China [8,13]. As smart devices becoming an indispensable part of human life, PD also inevitably need to use modern communication products. Smart technology has evolved the ways of interaction such as input becomes based on tiny touch screens, on screen keyboards, on gesture and finger input [1]. But the physiology and psychology of the patients with PD are very different from the target customers of smart phones [1,8,22]. Cardinal symptoms of PD like rigidity and tremor, neurological disorders like insomnia and severe fatigue, undermined speech and voice capabilities are likely to reduce the individual's mobility and autonomy and thus affect the way they interact with smart devices [8].

At present, the smart device with less functions can solve the basic needs of the special groups, but cannot meet the fashion need of the special groups for health, recreation and social demand [1]. The problems of existing smart devices and computers are divided into three sectors: user interface, interaction design, physical capabilities. As for user interface, decorative fonts prevent patients with visual problems reading text easily [5], patients with visual problems find it hard to adapt

to the brightness changes between screens [1,4], elder patients may have problems with perception of the application and the devices itself and verbiage in system requires an understanding of smart devices basics to make sense [10]. In terms of interaction design, for smart devices, the interaction is mainly implemented by gestures on the screen. And the problem lies in that motor-impaired PD patients cannot accurately tap on a small icon on the screen or perform those interaction gestures well, like sliding or scrolling. Buttons of touch screens react too fast and this is a problem for the elderly [4]. Audio interaction is also not feasible as PD has strong effects in patients' language competence, which result in their lack of clarity and loudness and make it hard for ASR to recognize what they are saying [22]. What's more, spreading information across screens or forcing scrolling may be problematic due to working memory decline [2] and decline in ability to coordinate multiple tasks using attention switching [3]. Screens which present too many options or buttons may become confusing due to patients scanning the whole screen towards their goal rather than zeroing in, by visual cues, on what they are looking for [10]. Considering physical capabilities, smartphones usual keypad is not good for elderly and the physical buttons from mobile phones are too small. Design of the size should consider the trade-off between the screen size and handset size for patients to hold. As for the volume of audio interaction, when some instructions of a UI want to be given by voice, patients with audio impairment cannot receive the signal in some frequencies range [1]. Consider whether to listen carefully to the elderly, but also need to consider the impact of the size of the sound on vision [12].

To provide an optimal design to the intended audience, we designed and developed PD-Helper to satisfy the patients' need of communication, entertainment and requirement for medical care. With consideration of usability, accessibility and customization, application is developed based on scanning technique to provide single touch interaction over entire screen of smart devices. Personalization feature allows patients to adjust the font size, background color and scanning speed according to one's individual pace. 40 users with PD, participated in the evaluation process of the prototype. Preliminary results are promising and indicate a good level of acceptance. The contributions of this study are that 1) an easy and feasible solution that helps PD patients to enhance their interaction with smart devices 2) A single platform that provides access to four major daily life utilities to satisfy the fundamental needs of PD patients 3) To accommodate the variant conditions

of PD Patients, they can choose controllable speed, and entire screen as touch keypad helps to improve the accuracy, single touch interaction smoothly fits to their physical limitations and helps to reduce physical fatigue. 4) This research also insights that by improving the accessibility and usability of interaction design of smart devices, they can be really helpful to provide a more productive, independent, social and fulfilling life to elder society.

II. RELATED WORK

Assistive technology is trying to assist the elderly society and motor impaired by providing a variety of applications to cope with interaction challenges in web search, text input as well as for health care. Keyboard is an essential modularity to interact with mobile phones. Researchers tried to optimize the QWERTY keyboard for a high efficiency by changing the layout [21], changing the key size, width or color [17,18] and different kinds of gestures such as swipe [19], shake [14] and other hand gestures [16] are used to promote typing efficiency. For web browsing, [11] developed a browser allowed patients to choose the links by presenting a series of brain responses within limited time. Web Accessibility Initiative provides bigger text with larger clicking areas and patients can order a special mouse to deal with the problem of shaking hands. For patients' Health care system, [7] did a survey to determine participants' perceptions of whether PROME would facilitate patient-physician communication. For multimedia, [23] created a Brain-Computer-Interaction system based on electroencephalography (EEG) signals to meet the recreation need of people with severe disability.

Though the development of the IME seems highly improved the efficiency of typing for healthy people, but the number of keys and the complexity of gestures increase difficulty for patients whose hands are shaky to tap or follow the gestures that is why systems [14-21] are not suitable for PD patients.

The browsers which have been developed [9, 11, 23, 24] have a strong limitation due to precise use of mouse control movement, hence is not adaptable to PD patients. It is more important for doctors to get patients feedback to decide whether the medicine is suitable or not [6,7]. But the need to access medical services directly by expressing patient feeling is still need to pay attention.

Multimedia and recreation content access is really helpful to change the mind state of patients especially when they feel depressed. PD patients are incapable to use most of them [23]. All above mentioned problems highlight the need of our proposed PD-Helper system that can help PD patients to interact with smart device applications according to their physical capabilities. The details are given below in the design section.

III. APPLICATION DESIGN

To make our design in accordance with PD patients' needs and physical capabilities, we have considered following aspects: *accessibility*, simple design without intervention of wearable sensors and additional hardware and interaction technique should be feasible for PD patients; *usability*, the system is easy to learn and use; *customization*, users are allowed to make adjustment to the system settings according to their own

controllability. We carefully considered and applied the three aspects in user interface design, interaction design and physical capabilities adaptation.

A. User interface design

- Main interface: PD-Helper provides four modules, namely website browsing, multimedia access, notebook and health care. As for the layout of the four modules and main interface, we adopted a neutral color scheme as background color and highlighting colors as icons' color, in order to ease the visual burden for PD patients to distinguish the flashing icons. The layout is concise and consistent in the whole system. Each icon and graphic representing modules or functions are easy to identify and are displayed with a larger size than that of the traditional mobile systems.
- IME interface: The layout is designed with 7 keys and one large size touch pad to make the interaction easier and efficient for motor impaired patients. First the initial alphabet (consonant) of pinyin is displayed. After that the system will show the possible pinyin combination. By selecting desiring combination, system will display the Chinese characters. The desired choices in each step are displayed within 7 keys.

B. Interaction design: Single touch and menu selection

PD-Helper provides a single touch interaction and whole screen as input area for users. Considering the trembling or rigidity symptoms, single touch allows users to touch the screen for a very short time to realize input and the application sets a large touch pad for them to touch and decrease the probability of error. Users do not click over individual keys and alphabets to select, they can tap over the entire area to make a selection. Menu selection is realized using scanning technique rather than touching on the small area of an icon. When the target items start to flash in turn, users can touch anywhere on the screen to select them.

C. Input technique: IME and shortcut menu

Consonant Character Input Method [25] is used in the IME application. To control the interaction via single touch, scanning ambiguous keyboard concept is used, because this is an efficient way to implement the system with reduced size and control the output via single key [23]. Sequence of typing one word is as follows. First look up for the initial alphabet (consonant) of pinyin. After that system will show the possible pinyin combination. By selecting desiring combination, system will display the Chinese characters. A small offline database connects the IME application to retrieve the Chinese characters efficiently. To maintain the efficiency during whole typing process, an online database is connected. So that if user does not find desired character, he just need to select the "x", system will type pinyin directly in the textbox. After user finished one sentence, system will replace pinyin with corresponding Chinese character.

D. Physical capabilities adaptation

A design should meet the patients' physical capabilities to make it adaptable at user level. Fig.1 shows the difference between device interaction and user capabilities to interact with

them. As interaction techniques based on precise target selection are out of questions for the patients, so it is really challenge for them to cope with these devices.

PD-Helper uses *scanning technique* to make selection, Icons flash in turn and users can touch on the entire screen to realize their input. *Single tap input* allows users to input with least physical effort. In this way, these applications accessible to more people with physical disabilities. *Personalization settings* are provided to change the scanning speed (4s, 3s, 2s), and color theme to facilitate the diverse conditions of PD. The web user controls with scanning technique and video easy control help patients to access the applications.

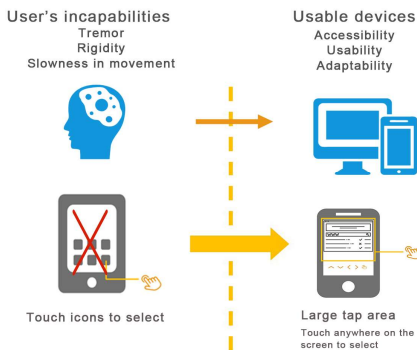


Figure 1. Physical capabilities adaptation

IV. APPLICATION IMPLEMENTATION

To implement the proposed design, we developed PD-Helper for Android platform. PD-Helper is composite of four modules, namely web browsing, multimedia, notebook and healthcare and provide patients with social connectivity, medical care and entertainment. The detailed implementation of each module in the application is listed below:

A. Main menu

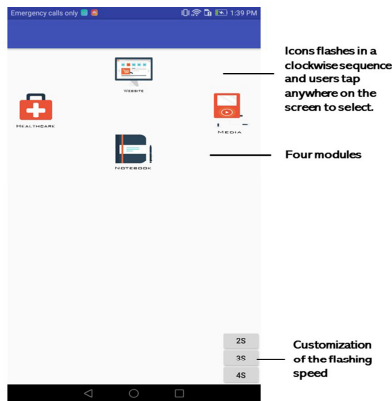


Figure 2. Main menu of PD-Helper

Main page displays four modules and their icons flash clockwise from website to multimedia, then notebook and the lastly healthcare (see Fig.2). Lower most right settings help to change the scan interval speed of the entire application. The

selection in PD-Helper can be made by single tap anywhere on the entire screen.

B. Web browsing

This module allows patients to browse the internet. When user opens browser, internet browser page divides into 2 parts: *page control area* and *web control area* (see Fig.3). *Web page control area* consists of web controls that start to scan in turn and helps to increase and decrease the font, refresh the current page, open new page and return to main menu by single tapping over the screen.



Figure 3. Web-Browsing PD-Helper

C. Multimedia

Video source: Multimedia Access provides a list of videos for users to watch, including some classic movies and rehabilitation treatment videos (see Fig.4).

Easy control: users touch the screen to select the video, and it plays automatically. Users can pause and play by single tapping for a short time, and then tap it again to continue to play. Touching the screen for a bit long time can exit the video and return to the video list. This mechanism is easy for PD patients to control videos.

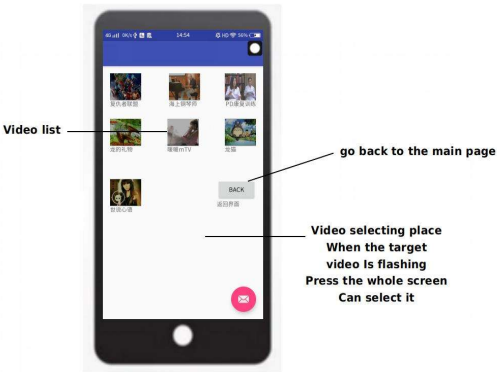


Figure 4. Multimedia module of PD-Helper

D. Notebook

This module provides an IME application that designed specifically for PD patients to type Chinese character in the text input area via single touch. Notebook application helps to type text, save notes, record notes and view the previous records. Users can also browse some relevant articles by typing keyword via “search” function. It provides many motivational cards with pictures and some motivational words to encourage users to change their mood. The content of the cards will be changed with tapping on the screen. Patients often have trouble in sleeping well due to physical therapy and psychological stress. The subsection “Sleep” provides many hypnotic music to improve PD patients’ sleep quality (see Fig.5).

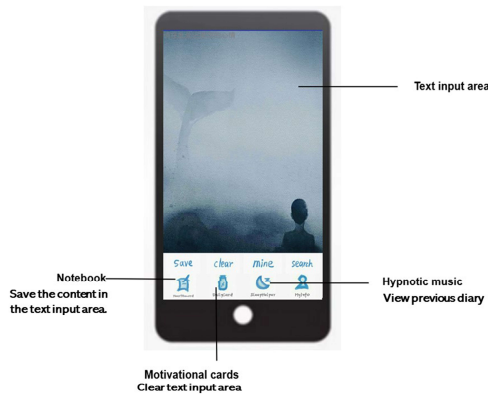


Figure 5. Notebook module of PD-Helper

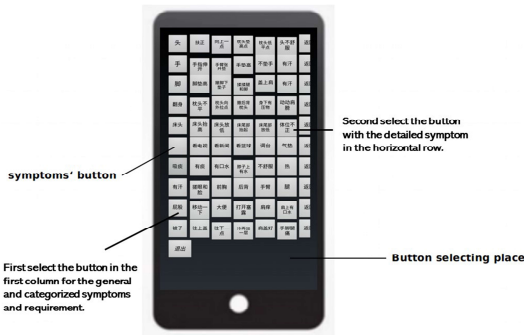


Figure 6. Health care connectivity

E. Healthcare

PD patients may suffer from speech impairment and are unable to express their need. This module provides common possible symptoms and needs for patients to express their feelings to the medical care staff automatically via the message system (see Fig. 6). The system helps to select the category of symptoms from the column via scanning technique and then pick the detailed symptoms in the corresponding row via single tap over the entire screen. The message will be sent to the contact number of care taker staff.

V. USER STUDIES

We conducted user studies to test the usability and to determine the effects of PD-Helper with PD patients. 40

recruited patients (18 males, 22 females) with age ranging from 43 to 89 (mean=69; SD=11.03) have been divided into two group according to their PD symptoms, namely tremor (T) and rigidity(R). Neurologist helped us to make selection of patients based on information from hospital records and nursing staff. Patient inclusion criteria includes 1) Parkinson’s disease symptoms (Rigidity, Tremor), 2) off state. Neurologist helped to motivate the patients to take part in the user study voluntarily. Neurologists helped to collect UDPRS data of participants. We divided the patient’s disease degree of severity into 3 levels only (using UDPRD data). All the patients signed informed patient consent form willingly and voluntarily participated to test the system.

To test the adaptability and accessibility of PD-helper with patients an experiment with 30 min time span is conducted. Keeping in view patient physical condition, it was tiring for a patient to test all the modules of PD helper. To test each module, we have divided patients into groups based on their capability to test the different modules. 22 Patients who were familiar with pinyin helped to test the notebook /IME module. The 18 patients with no or little knowledge of smart devices helped to test the web browsing and multimedia module. During the test sessions, participants had to perform the following tasks with three different scanning speeds.

Task 1: play the third video from the multimedia module, and check the play, pause and close functions.

Task 2: Patient has to open SOHU website and used the all available web user controls to increase, decrease font, refresh the page, open new page, and back to main menu.

Task3: Type a random sentence (selected from the old Chinese poem and daily life sentences) with the note book module (see Table III).

Test was conducted on Tablet (7.5 inches screen, android OS). Patients performed the test while holding the tablet in assistant hand, in their own hand and over the table as well.

A. Results and data of Task 1

18 participants took part to complete the video test. During test session, patient has to play a video, then use play, pause and close features of the application that are specifically designed to assist the diverse conditions of PD. The aim of the test is to measure the accuracy rate of users’ operation with different speed interval on video task. Before the test, we estimated the time required to complete the task without error and the estimated time (ET) for 2-4s speed interval is respectively 71.6s, 73.6s and 75.2s. We then measured the actual time (AT) participants took to complete the task in the test. Accuracy rate calculated in the following way:

$$R (\text{Accuracy rate}) = 1 - 100\% * (AT - ET) / AT \quad (1)$$

TABLE I. ACCURACY AND ERROR RATE OF TASK 1

Speed Interval	Accuracy Rate		Error Rate	
	T	R	T	R
4s	95.4	91.9	4.6	8.1
3s	88.0	89.9	12.0	10.1

Speed Interval	Accuracy Rate		Error Rate	
	<i>T</i>	<i>R</i>	<i>T</i>	<i>R</i>
2s	78.7	80.5	11.3	19.5

B. Results and data of Task 2

18 participants took test in web browsing module and all of them have completed the test. The test aims at testing the usability of web browsing mechanism and measuring the accuracy rate of users' operation with different speed interval. Before the test, we estimated the time required to complete the task without error and the estimated time (ET) for 2-4s speed interval is respectively 30s, 40s and 50s. We then measured the actual time (AT) participants took to complete the task in the test. Accuracy rate calculated by equation (1).

TABLE II. ACCURACY AND ERROR RATE OF TASK 2

Speed Interval	Accuracy Rate		Error Rate	
	<i>T</i>	<i>R</i>	<i>T</i>	<i>R</i>
4s	92.2	98.7	7.8	1.3
3s	85.3	90.4	14.7	9.6
2s	77.6	79.9	22.4	20.1

C. Results and data of Task 3

22 Participants overall have been divided according to their degree of severity to test the correlation between degree of severity, different speed interval (1s, 2s, 3s) and typing speed.

Patient used Table III sentences to complete the task 3. By keeping in view patients' physical condition short random sentences are preferably selected to avoid from tiredness and physical fatigue. In 1s speed interval test, the numbers of people with degree of severity from 1 to 3 are respectively 4, 13, 5. In 2s speed interval test, there are 19 people with 2 degrees of severity and 3 people with 3 degrees of severity. In 3s speed interval test, 10 people are with 2 degrees of severity and 12 people are with 3 degrees of severity.

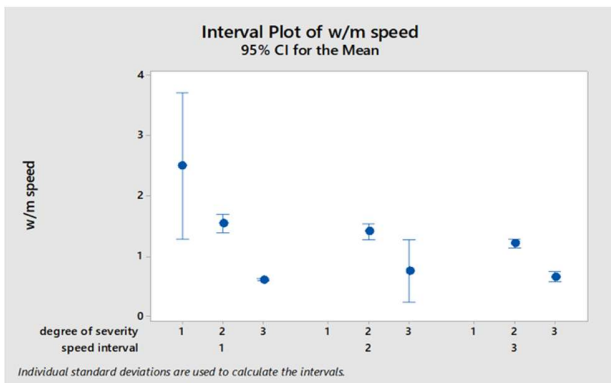


Figure 7. Interval Plot of w/m speed

In terms of speed interval in relation to typing speed (Fig. 7), people have an approximately average typing speed of 2

words/minute with 1s interval, 1.1 words/minute with 2s interval, 0.65 word/minute with 3s interval. There's negative linear relationship between degree of severity and speed of typing. People have an approximately average typing speed of 2.3 words/minute with 1 degree of severity, 1.5 words/minute with 2-degree of severity, 0.7 word/minute with 3-degree of severity.

TABLE III. TEST SENTENCES FOR TASK 3

Sentence	Pinyin	Meaning
谢谢	xie xie	Thanks
没问题	Mei wen ti	No problem
我在北京	Wo zai bei jing	I'm in Beijing.
妈妈有三个孩子	Ma ma you san ge hai zi	Mom has three children
欢迎使用这个系统	Huan ying shi yong zhe ge xi tong.	Welcome to use this system

VI. DISCUSSION

From above data results section, it is proved that single touch PD-Helper is effective for patients to access the four major daily life utilities with more ease and accuracy via smart devices. The reason was that touch interaction technique was based on user experience and it was easy for the patients to adapt the one touch interaction technique as they were already familiar with it rather than learning novice and complex gestures. All patients used the three flash speeds to complete the test irrespective of their severity. During task 1 tremor patients showed more accuracy 95% with 4s as compared to rigidity patients 91%. During task 2 the lowest rate for accuracy was 77 % and 79 % of tremor and rigidity respectively. That shows really good acceptance especially in case of PD patients. It showed that patients adapt only those features that are in accordance with their capabilities. User's questionnaire feedback shows that IME is easy to use due to simple consonant vowel structure and personalized features make it suitable for diverse severity patients. The proposed system is best combination of accessibility, accuracy and usability.

Accessibility: Different scanning speed level can be adjustable according to patient physical condition, so patients with diverse conditions can access PD-Helper without any discomfort

Accuracy: large size icons, font size and color theme provide more clarity and visibility to the patients. A fatigue of exact target selection is also eliminated. To make selection of target they can touch anywhere of the screen. It helped to increase the accuracy for patients.

Usability: single tap selection along with large touch area reduce their motoric interaction and hence physical fatigue for them.

Fig.8 shows the user study performed with patients. This user study has made major relevant contributions in minimizing the motoric interaction for PD patients to access the smart devices. The idea of PD-Helper via single tap interaction over smart

devices is introduced for the first time for PD patients that is why the comparative study was not possible. All users appreciate the idea and practical worth of this system



Figure 8. User Study with PD-Helper Patients in Xuan Wu Hospital

VII. CONCLUSION

Parkinson's disease (PD) Patients with diverse physical conditions exhibit challenges in interacting with smart devices due to motor impairment. To enhance their interaction, and accessibility with smart devices, PD-Helper is designed and developed with single touch interaction as well as a large touch pad area to improve tactile experience for intended audience. Scanning technique is used to control input via single tap to minimize the physical fatigue for patients. By virtue of personalization features 40 participants (PD patients) were able to access the web, multimedia content and input Chinese text with more accuracy. The IME application, Web user controls, video easy controls, and connectivity with health care staff helps PD patient to improve their social connectivity via smart devices. Further studies are in progress to attest the PD helper potential, such as improving the quality of life of people with other Neurological disorders. Current limitation of the system is that it is developed for Chinese language. As a future work, a multilingual PD-Helper solution will be offered to support the patients around the globe.

ACKNOWLEDGEMENT

The project is supported by the Project: 61672017 of National Science Foundation of China.

REFERENCES

- [1] Carmien S., Manzanares A.G. (2014) Elders Using Smartphones – A Set of Research Based Heuristic Guidelines for Designers. In: Stephanidis C., Antona M. (eds) Universal Access in Human-Computer Interaction. Universal Access to Information and Knowledge. UAHCI 2014. Lecture Notes in Computer Science, vol 8514. Springer, Cham
- [2] Fisk, A.D., Rogers, W. A., Charness, N., Czaja, S. J., & Sharit, J. , Designing for Older Adults: Principles and Creative Human Factors Approaches 2004: CRC Press J Taylor&Francis Croup
- [3] Jacko JA, Barreto AB, Marrnet GJ, Chu JYM, Bautsch HS, Scott IU, Rosa RH (2000) Low vision: the role of visual acuity in the efficiency of cursor movement. In: Proceedings of ASSETS'00, Arlington, VA, November 2000. ACM Press, pp 1–8
- [4] Strengers, J., Smartphone interface design requirements for seniors, in Information Studies 2012, University of Amsterdam: Amsterdam.
- [5] Kurniawan, S. and P. Zaphiris, Research-derived web design guidelines for older people, in Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility 2005, ACM: Baltimore, MD, USA. p. 129-135.
- [6] Krpic, Andrej et al. "Telerehabilitation: remote multimedia-supported assistance and mobile monitoring of balance training outcomes can facilitate the clinical staff's effort." *International journal of rehabilitation research. Internationale Zeitschrift für Rehabilitationsforschung. Revue internationale de recherches de readaptation* 36 2 (2013): 162-71 .
- [7] Hong, Song Hee et al. "Health Care Applicability of a Patient-Centric Web Portal for Patients' Medication Experience." *Journal of medical Internet research* (2016).
- [8] Francisco Nunes • Paula Alexandra Silva • Joaõ Cevada • Ana Correia Barros • Lui 's Teixeira, User interface design guidelines for smartphone applications for people with Parkinson's disease, *Univ Access Inf Soc* (2016) 15:659–679 DOI 10.1007/s10209-015-0440-1
- [9] Hanson, Vicki L. and Susan Crayne. "Personalization of Web browsing: adaptations to meet the needs of older adults." *Universal Access in the Information Society* 4 (2005): 46-58.
- [10] Chisnell, D., Redish, J. Designing web sites for older adults: a review of recent research. AARP.org/olderwisewired, 2004.
- [11] Bensch, Michael et al. "Nessi: An EEG-Controlled Web Browser for Severely Paralyzed Patients." *Computational Intelligence and Neuroscience* 2007 (2007): 297 - 298.
- [12] Minggang Yang and He Huang, Research on Interaction Design of Intelligent Mobile Phone for the Elderly Based on the User Experience
- [13] Rodrigues, Élvio, Micael Carreira, and Daniel Gonçalves. "Developing a multimodal interface for the elderly." *Procedia computer science* 27 (2014): 359-368.
- [14] Mark Dunlop, Andreas Komninos, Emma Nicol, and Iain Hamilton. 2014. Shake 'n' Tap: a gesture enhanced keyboard for older adults. In Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services (MobileHCI '14). ACM, New York, NY, USA, 525-530.
- [15] Hutchison, Elizabeth A et al. "Diversification of a protein kinase cascade: IME-2 is involved in nonself recognition and programmed cell death in *Neurospora crassa*" *Genetics* vol. 192,2 (2012): 467-82.
- [16] Yin, Ying et al. "Making touchscreen keyboards adaptive to keys, hand postures, and individuals: a hierarchical spatial backoff model approach." *CHI* (2013).
- [17] Hsiao HC, Wu FG, Chen CH. Design and evaluation of small, linear QWERTY keyboards. *Appl Ergon*. 2014 May;45(3):655-62. doi: 10.1016/j.apergo.2013.09.001. Epub 2013 Sep 26. PubMed PMID: 24075287.
- [18] Antti Oulasvirta, Anna Reichel, Wenbin Li, Yan Zhang, Myroslav Bachynskyi, Keith Vertanen, and Per Ola Kristensson. 2013. Improving two-thumb text entry on touchscreen devices. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13). ACM, New York, NY, USA, 2765-2774.
- [19] Sakkos, Panos et al. "Anima: Adaptive Personalized Software Keyboard." *CoRR* abs/1501.05696 (2015): n. pag.
- [20] Bi, Xiaojun et al. "Multilingual Touchscreen Keyboard Design and Optimization." *Human-Computer Interaction* 27 (2012): 352-382.
- [21] Bi, Xiaojun, Barton A. Smith and Shumin Zhai. "Quasi-qwerty soft keyboard optimization." *CHI* (2010)
- [22] Ondrej Polacek1 • Adam J. Sporka1 • Pavel Slavik1, "Text input for motor-impaired people", *Univ Access Inf Soc* (2017) 16:51–72, DOI 10.1007/s10209-015-0433-0
- [23] Jabeen, Farzana et al. "Mind interactive multimedia system for disabled people." 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI) (2017): 1-6.
- [24] Poulson, David and Colette Nicolle. "Making the Internet accessible for people with cognitive and communication impairments." *Universal Access in the Information Society* 3 (2003): 48-56.
- [25] Jabeen, Farzana, Tao, Linmi, Wang, Xinyue, et al. C-SAK: Chinese Scanning Ambiguous Keyboard for Parkinson's Disease Patients. In : 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech). IEEE, 2018. p. 792-799.

ACNET: Attention-based Convolution Network with Additional Discriminative Features for DCM Classification

Chao Luo

*College of Computer Science
Chengdu University of
Information Technology
Chengdu, China
clchaoluo@163.com*

Wang Xin

*College of Computer Science
Chengdu University of
Information Technology
Chengdu, China
xinw@ciracloudcorp.com*

Xiaojie Li*

*College of Computer Science
Chengdu University of
Information Technology
Chengdu, China
lixiaojie000000@163.com*

Yucheng Chen*

*West China Hospital
Sichuan University
Chengdu, China
chenyucheng2003@126.com*

Jiliu Zhou

*College of Computer Science
Chengdu University of
Information Technology
Chengdu, China
zhoujiliu@cuit.edu.cn*

Kunlin Cao & Youbing Yin

*CuraCloud Corporation
Seattle, USA
cao, yin@ciracloudcorp.com*

Qi Song

*CuraCloud Corporation
Seattle, USA
song@ciracloudcorp.com*

Xi Wu

*College of Computer Science
Chengdu University of
Information Technology
Chengdu, China
xi.wu@cuit.edu.cn*

Abstract—For dilated cardiomyopathy (DCM) patients, immediate emergency diagnosis and treatment are critical for life saving and later recovery. T1 mapping is a non-invasive and effective diagnostic imaging approach to detect DCM. However, it is a demanding and time-consuming approach. In this paper, we propose an attention-based network structure, which can automatically identify DCM patients in a speedy manner to prioritize their treatment. In the proposed method, we adopt attention modules to generate attention-aware features. Inside each attention module, a bottom-up top-down feed-forward structure is used to unfold the feed-forward and feed-back attention processes into a single feed-forward process. It allows the network to focus more on determining useful information about the current output that is significant in the input data. Moreover, inspired by the residual network idea, we make full use of the characteristics of the original data. Combined residual block, we design down-residual modules for classification tasks. It consists of seven convolution layers and three layers of residual blocks. Our network achieves the most advanced recognition performance on cardiac datasets. We evaluated our approach on CMR(cardiac magnetic resonance) T1 mapping images with lower PSNR(peak signal to noise ratio), and the results demonstrate that our architecture outperforms previous approaches.

Index Terms—medical image classification, attention, classification, myocardial

I. INTRODUCTION

Image classification is one of the most important tasks in computer vision tasks [1], and it is also the basis of other high-level visual tasks such as object tracking and behavior analysis [2] [3]. Particularly in medical image tasks, medical image classification is the key issue to determining whether medical images can provide a reliable basis for clinical diagnosis and treatment [4]. The development of medical classification technology plays an extremely important role in biomedical image analysis. In recent years, classification technology has made significant progress because of the application of deep learning algorithms in medical image classification. This is more challenging than classification tasks on natural images. The highly cluttered background and the particularity of medical images pose major challenges to classification accuracy [5] [6].

Dilated cardiomyopathy (DCM) is a common myocardial disease [7] [8] [9]. The disease can lead to ventricular systolic dysfunction, congestive heart failure and arrhythmia. The disease is progressively aggravated, and death can occur at any stage of the disease. Therefore, the use of DCM data as a dataset is of great significance [10] [11] [12] [13].

At present, many methods use medical image classification, such as, the SVM-based(Support Vector Machine) classification method [14] [15] and texture-based method [16] [17]. However, on comparing various methods, the method based on an artificial neural network achieves better results. It is found that the artificial neural network has learning and recognition abilities similar to the human brain. Moreover, it can model human organs, and independently learn the determined

* These authors are co-corresponding authors. This work was supported by the National Natural Science Foundation of China (Grant No. 61602066) and the Scientific Research Foundation (KYTZ201608) of CUIT and the major Project of Education Department in Sichuan (17ZA0063 and 2017JQ0030), and partially supported by the Sichuan international science and technology cooperation and exchange research program (2016HH0018). DOI reference number: 10.18293/SEKE2019-155

diagnostic information and diseased tissue. Additionally, it can improve the reliability and effectiveness of diagnosing a patient's condition. Therefore, in this paper, we use an attention-based neural network method to effectively classify medical images.

Inspired by the residual attention network and recent advances in deep neural networks [18] [19], we propose a new network based on attention. It is composed of two attention modules that generate attention-aware features. Additionally, we design the down-residual module. It is capable of extracting the high-level features of the input data without causing a loss of the original features. As the number of layers increases, the attention-aware features from different modules are adaptively changed.

In addition to the additional discriminating features acquired by the attention module, our network has the following advantages: 1) Before the attention module, we design a down-residual module, which enables the extraction of high-level features of the input data without causing a loss of the original features. 2) Because of the scalability of the attention module, our network can be easily extended to hundreds of layers. Our network outperforms state-of-the-art residual attention networks. Experimental results show that the performance of our proposed network is better than that of residual attention network.

II. ATTENTION MODULE

The visual attention mechanism is a brain signal processing mechanism that is unique to human vision [20] [21]. Human vision scans the global image quickly to identify the focus area, which is generally called the focus of attention, and then invests a large amount of attention resources in this area to obtain more details of the target and suppress other useless information. The attention mechanism in deep learning is essentially similar to the human selective visual attention mechanism. The core goal is to also select more information from the many information that is of vital importance to the current mission objectives [22]. In recent years, the attention model has been widely used in various types of deep learning tasks, such as natural language processing, image recognition and speech recognition [23] [24] [18]. It is a core technologies that deserves the most attention and insight. In this paper, the attention model used is based on the attention module of residual attention network.

To improve the classification accuracy, Wang fei et al. proposed a residual attention network [25]. Compared with ResNet-200 [26], the residual attention net achieves 0.6% top-1 accuracy improvement and has good robustness. Based on the above advantages, we adopt an attention module in a residual attention network. As shown in Figure 1, each attention module is divided into two branches: the mask branch and trunk branch. The trunk branch performs feature processing and can be adapted to any state-of-the-art network structure. For the mask, first, through a series of convolution and pooling, it gradually extracts high-level features and increases the receptive field of the model. Then the size of

the feature map is enlarged to the same size as the original input by the same number of up samples. Thus, it maps the area of attention to each pixel.

Given input x , $T(x)$ denotes the trunk branch output, and the mask branch uses a bottom-up top-down structure to learn the same size of mask $M(x)$ that softly weights output features $T(x)$. The output mask is used as control gate for neurons of the trunk branch that are similar to the highway network. The output of attention module $H(x)$ is:

$$H_{i,c}(x) = (1 + M_{i,c}(x)) * T_{i,c}(x), \quad (1)$$

where i ranges over all spatial positions, c is the index of the channel ($c \in \{1, \dots, c\}$), $M(x)$ is the output of Soft Mask Branch and $T(x)$ is the output of the Trunk Branch. The entire structure can be trained end-to-end. Each pixel value in the attention map output by the mask branch is equivalent to the weight of each pixel value in the original feature map, which enhances meaningful features and suppresses meaningless information. Therefore, the weighted attention map is obtained by element-wise multiplication of the output of the mask branch and the output of the trunk branch. However, this weighted attention map cannot be directly input into the next layer because the activation function of the mask branch is sigmoid and the output value is in the range $(0, 1)$, so the author performs an element-wise operation on the weighted attention map and the feature map of the trunk branch.

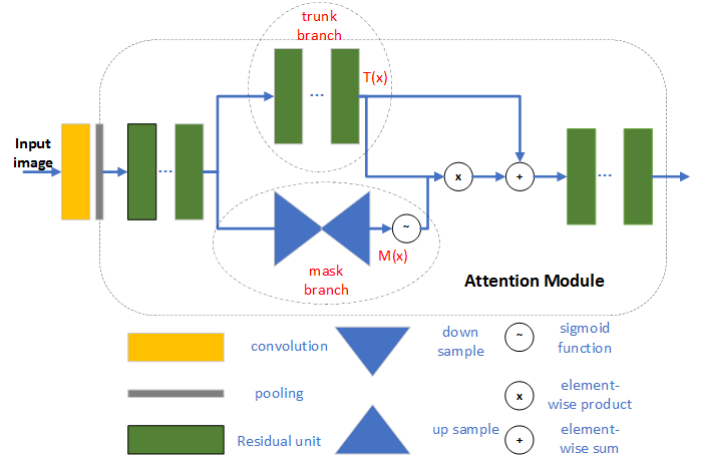


Fig. 1. Network structure of the attention module.

The attention module has the following advantages: 1) The attention module is similar to the residual learning mode, so a very deep model can be easily optimized and learned, and has very good performance. 2) The forward attention mechanism of bottom-up top-down. Other networks that use attention often need to add a branch to the original network to extract attention, and the author's model can extract the attention of the model in a forward process, thereby making the model training easier.

III. METHOD

In this paper, we construct a down-residual module and combine it with the attention module to construct an attention-based convolution network(ACNet). The network combines the advantages of the down-residual module with the advantages of the attention module to deliver outstanding performance and robustness.

A. Attention-based Network

We construct an attention-based convolution network based on attention. The overall network architecture and hyperparameter settings are shown in Figure 2. The network consists of three modules: the first module consists of seven convolutional layers and three residual blocks, and then two attention modules are connected. The former operation quickly collects the global information of the entire image and the latter operation extracts high-level distinguishable feature information.

From input x , convolution are firstly performed several times to increase the receptive field rapidly. After achieving the high-level features, they are used as input into the three residual layers. The residual layer can fully extract the useful advanced feature information and combine it with the input information of the upper layer. The useful feature information is fully used and missing useful feature information is avoided. Finally, the features are put into two attention layers.

B. Loss Function

Cross entropy describes the distance between two probability distributions. The smaller the cross entropy, the closer the two are. It is often used for the classification loss function of deep neural networks. In this paper, we use cross entropy as the loss function of the myocardial classification.

$$H(y, y') = - \sum_i^n y'_i \log y_i, \quad (2)$$

where H is a representation of the loss value, y indicates the predicted probability distribution, y' denotes the true probability, and n suggests the number of samples. Furthermore, the cross entropy loss function is put to a frequent use in classification networks.

Learning rate strategy: We conducted several trials with different learning rates, and the results showed that the learning rate of 0.001 was the most appropriate. If the learning rate is too high, it will lead to over-fitting, and the network cannot learn the feature information correctly. If the learning rate is too low, the network fitting speed is very slow. Therefore we set the learning rate to 0.001 .

Experiment configurations: To ensure the consistency of the experiment, we use accuracy as the quantization metric, 100 epochs are trained for each experiment ($batch_size = 3$). All experiments are implemented in python2.7 by using Tensorflow and Keras framework. We train the networks on a NVIDIA Tesla M40 GPU and the model that performs the best on test data set are saved for further analysis.

IV. EXPERIMENT AND RESULTS

A. Dataset, Pre-processing and Evaluation Metrics

CMR T1-mapping images of 485 myocardial sections from the same hospital were both trained and tested. The ground truth of the training and the testing samples of images with the myocardial were manually annotated by an experienced cardiologist. The pixel size of each CMR image amounts was 1.406×1.406 mm with a size ranging between 192×256 and 224×256 . Considering the fact that the size of the myocardium is small and surrounded by a substantial amount of noise, we first resampled the resolution of the image to $1 \times 1 \times 1$, and each image was cut to a fixed image size of 128×128 . Finally, the intensity range of the T1-mapping images was normalized to $[0 - 255]$.

To improve the classification performance of ACNet, three groups of experiments were constructed. In experiment 1, we increased the dataset size by using 485 images with a size of 128×128 (included 36 normal people and 449 patients), 388 training images, and 97 test images. In experiment 2, because the ratio of the positive and negative samples of the above dataset was very unbalanced (the negative sample contained only 36 cases), the dataset may have caused the classification performance to decrease. To solve this problem, we used data enhancement methods such as image rotation and random increase of noise to increase the number of samples of normal people from 36 cases to 191 cases, a total of 640 datasets of size 128×128 [27]. A residual attention network, which is a broadly applied state-of-the-art network structure, was used as a baseline method. To conduct a fair comparison, we used most of the settings same as Residual Attention Network paper. We adopted the same weight initialization method as the previous study and trained residual attention network using nesterov SGD with a mini-batch size of 3.

We used four indicators to assess the performance of the network, which includes accuracy, sensitivity and specificity. The accuracy is the average of the accuracy of in each group of experiments. In the medical field, sensitivity is the probability of correctly predicting positive samples, and specificity is the probability of correctly predicting negative samples. Sensitivity is computed using a closed-form formula:

$$sensitivity = \frac{A}{A + B}, \quad (3)$$

where A is the number of positive samples that are correctly predicted and B is the number of positive samples with incorrect prediction results. Similarly, specificity is computed using a closed-form formula:

$$specificity = \frac{D}{C + D}, \quad (4)$$

where D is the number of negative samples that are correctly predicted and C is the number of negative samples with incorrect prediction results. Generally, the higher the above four indicators, the better the performance of the experimental method.

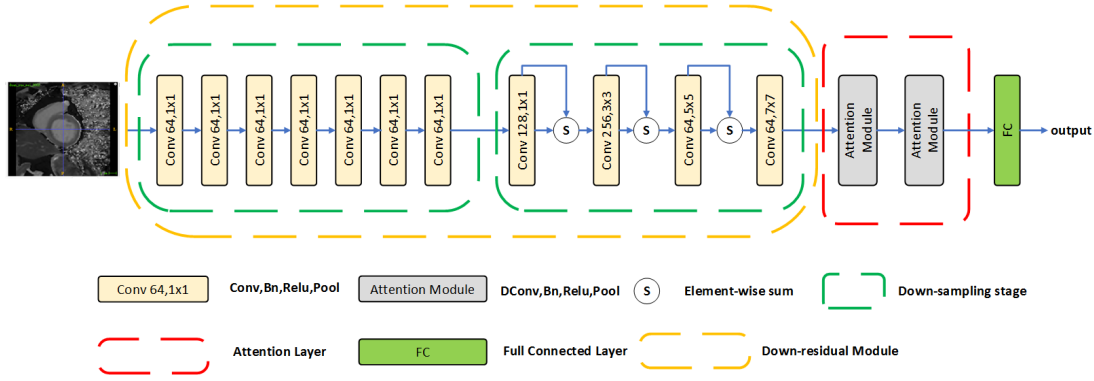


Fig. 2. The attention-based network structure proposed in this paper.

B. Results and Discussion

In this paper, we evaluated the performance of the residual attention network, VGG16, ResNet50, ResNet101 and our proposed network on the myocardial dataset. Two groups of experiments were constructed, and each group experiment was divided into two parts. In the first part, we used the myocardial dataset to perform multiple experiments on the residual attention network, VGG16, ResNet50 and ResNet101 to observe changes in the indicator. In the second part, we performed multiple experiments on the proposed attention-based network(ACNet) using the myocardial dataset to obtain the values of each indicator. Finally, the accuracy, sensitivity and specificity were compared to obtain the performance difference between the two methods.

For experiment 1, as shown in Table I, in the case of the same amount of myocardial data, the classification accuracy of the residual attention network was only 0.4179, the accuracy of VGG16 was only 0.8593, the accuracy of resnet50 is only 0.6251, and the accuracy of resnet101 is only 0.8347, however, the accuracy of our proposed network can reached 0.9277. Our approach is far superior to the other four comparison networks in terms of accuracy. Additionally, for the two indicators of sensitivity and specificity, our proposed network was also superior to the residual attention network, VGG16, ResNet50 and ResNet101. Unfortunately, the sensitivity of all methods in Table I was very low, whereas the specificity was close to 1. This phenomenon was caused by the imbalance between the number of positive samples and the number of negative samples. Therefore, to solve this problem in experiment 2, we used the data enhancement method to increase the number of negative samples, so that the difference between the positive and negative samples was reduced.

Figure 3 shows the attention features of two different samples in the residual attention network and ACNet. The myocardial instance mask highlighting the area surrounding of the myocardial. As can be observed from the figure, the myocardial mask highlighted the area around the myocardium and inhibited the middle area. The surrounding area of the myocardium is just the main feature area for assessing whether the heart muscle is normal. Therefore, we can know that the

TABLE I

EXPERIMENT 1 RESULTS FOR BOTH NETWORKS. IN THIS EXPERIMENT, A 5-FOLD CROSS-VALIDATION METHOD WAS USED. THE AVERAGE ACCURACY WAS THE AVERAGE OF FIVE EXPERIMENTAL PRECISIONS. THE SENSITIVITY AND SPECIFICITY WERE THE AVERAGE RESULTS OF FIVE EXPERIMENTS.

Network	Accuracy	sensitivity	specificity
Residual attention network	0.4179	0.1067	0.9571
VGG16	0.8593	0.3642	0.9142
ResNet50	0.6251	0.1753	0.8327
ResNet101	0.8347	0.3162	0.9059
Ours	0.9277	0.5305	0.9901

attention mechanism can highlighted useful feature information while suppressing the useless feature information, thereby improving the classification performance. Additionally, by comparing the feature maps of the two methods, we can clearly observe that ACNet's ability to highlight useful information and the ability to suppress useless information was stronger than that of the residual attention network. It essentially shows that ACNet outperformed the residual attention network.

For experiment 2, as shown in Table II, we used the data enhancement method to increase the number of negative samples. This method can reduce the difference in the number of between positive and negative samples, the number of positive samples and the number of negative samples remained relatively balanced. From Table II, we can observe that ACNet's three indicators were much higher than those of residual attention network, VGG16, ResNet50 and ResNet101. This result once again demonstrates that the performance of our proposed method is the most outstanding.

Figure 4 shows the comparison of the accuracy results of the residual attention network, VGG16, ResNet50, ResNet101 and ours ACNet. From the figure, we can clearly observe that, in the case of the same DCM dataset, the ACNet classification accuracy was always much higher than that of remaining four networks. From Figure 5, we can observe that the specificity of ACNet was much larger than that of remaining four networks, indicating that ACNet's ability to predict correct negative

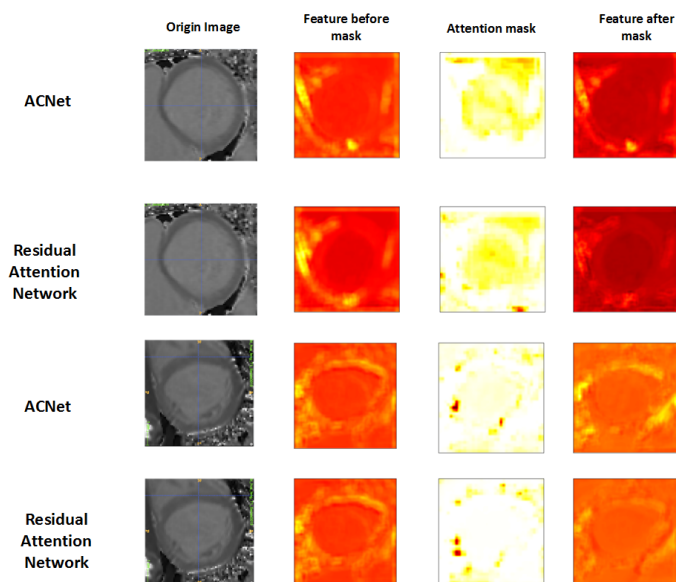


Fig. 3. Example images illustrating that different features have different corresponding attention masks in attention module. The myocardium instance mask highlights high-level myocardium surrounding part features.

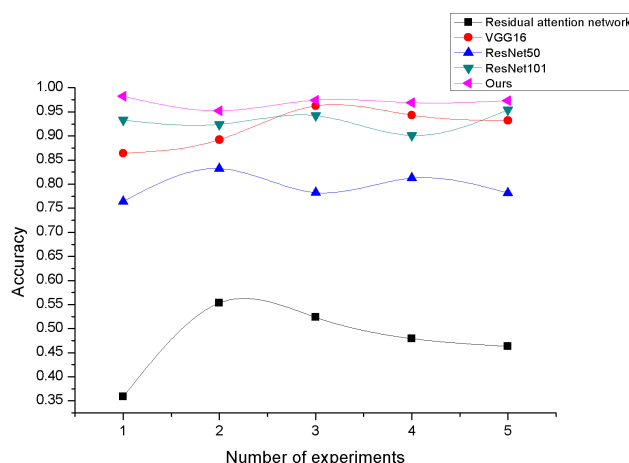


Fig. 4. Distribution of accuracy results in experiment 2.

samples was significantly higher than that of remaining four networks. The difference in sensitivity results between the residual attention network, VGG16, ResNet50, ResNet101 and ACNet is shown in Figure 6. From the figure we can observe that the sensitivity result of the residual attention network, VGG16, ResNet50 and ResNet101 was much lower than that of ACNet. This demonstrates that the ability of remaining four comparison networks to predict correct positive samples was far lower than that of ACNet.

In general, from a comprehensive analysis of the results of the above methods, we can clearly observe that the advantages

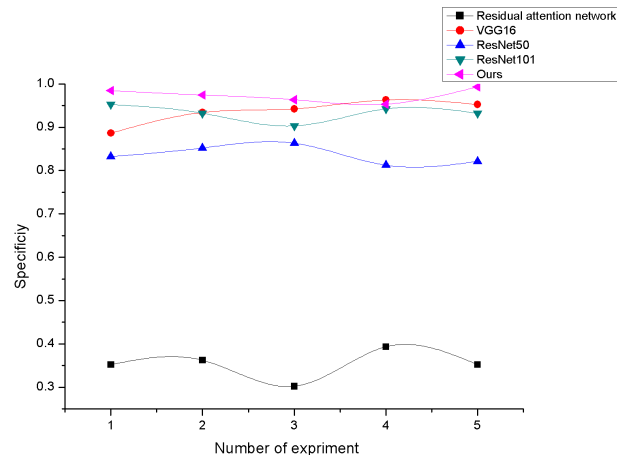


Fig. 5. Distribution of specificity results in experiment 2.

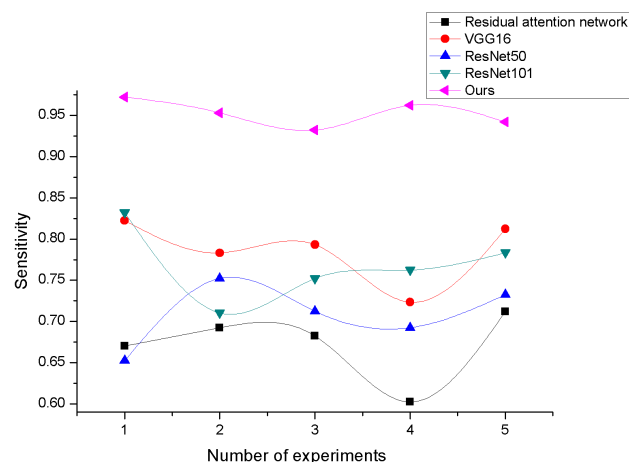


Fig. 6. Distribution of sensitivity results in experiment 2.

of ACNet proposed in this paper were very prominent. Our method is much higher than the four comparison methods in terms of accuracy, sensitivity and specificity. From Figure 4 6 5, we can see that the three indicators of our method are very stable. It can be seen that our method is very robust. In the medical image classification task, the performance of ACNet was far stronger than that of the residual attention network, VGG16, ResNet50 and ResNet101. From the many experimental results, we can observe that the experimental results of ACNet were very prominent and stable, the robustness of ACNet was also strong.

To summarize, in the two groups of experiments, the four indicators of ACNet were much larger than those of the remaining four comparison networks. Thus, we can clearly observe that ACNet outperformed the residual attention network, VGG16, ResNet50 and ResNet101. Furthermore, the three indicators of experiment 2 were much higher than those of experiment 1, in particular, the results of ACNet in experiment

TABLE II

EXPERIMENT 2 RESULTS OF BOTH NETWORKS. IN THIS EXPERIMENT, THE UP-SAMPLING METHOD WAS USED TO INCREASE THE NUMBER OF NEGATIVE SAMPLES.

Network	Accuracy	Sensitivity	Specificity
Residual attention network	0.4718	0.6890	0.3510
VGG16	0.9452	0.7891	0.9405
ResNet50	0.7859	0.7132	0.8327
ResNet101	0.9274	0.7672	0.9361
Ours	0.97152	0.9582	0.9761

2 were significantly higher than the results of experiment 1. Therefore, we found that after using the data enhance method, performance was greatly improved. Similarly, in the field of medical imaging, there are often problems such as small datasets, and unbalanced proportions of positive and negative samples. We can use the method of data enhance to appropriately increase the number of samples, so that the number of positive and negative samples is basically the same. This can greatly improve the classification performance of the network.

V. CONCLUSION

In this paper, we proposed an attention-based convolution network. It combines attention module and down-residual module designed that we designed. The down-residual module can acquire high-level features without losing the original features, and the attention module highlights useful features and suppresses useless features. To prove the effectiveness of our framework, we compared it with residual attention network, VGG16, ResNet50 and ResNet101. We found that the proposed method was superior to remaining four comparison networks in terms of three indicators. The experimental results show that this method is obviously superior to the most advanced classification methods, with the highest accuracy, sensitivity and specificity. Additionally, the framework that we used is generally applicable to the classification tasks of other medical images or natural images. We plan to conduct further research in the future.

REFERENCES

- [1] C. Tian, *A Computer Vision-Based Classification Method for Pearl Quality Assessment*, 2009.
- [2] N. Doulamis and A. Doulamis, "Semi-supervised deep learning for object tracking and classification," in *IEEE International Conference on Image Processing*, 2015, pp. 848–852.
- [3] S. Pang, J. J. D. Coz, Z. Yu, O. Luaces, and J. Diez, *Combining Deep Learning and Preference Learning for Object Tracking*. Springer International Publishing, 2016.
- [4] F. L. C. D. Santos, M. Paci, L. Nanni, S. Brahnam, and J. Hyttinen, "Computer vision for virus image classification," *Biosystems Engineering*, vol. 138, pp. 11–22, 2015.
- [5] T. M. Lehmann, H. Schubert, D. Keyers, M. Kohnen, and B. B. Wein, "The irma code for unique classification of medical images," in *Medical Imaging*, 2003.
- [6] I. Buciu and A. Gacsadi, "Gabor wavelet based features for medical image analysis and classification," in *International Symposium on Applied Sciences in Biomedical and Communication Technologies*, 2009, pp. 1–4.
- [7] C. Grefkes, S. B. Eickhoff, D. A. Nowak, M. Dafotakis, and G. R. Fink, "Dynamic intra- and interhemispheric interactions during unilateral and bilateral hand movements assessed with fmri and dcm," *Neuroimage*, vol. 41, no. 4, pp. 1382–1394, 2008.
- [8] N. K. Lakdawala, L. Dellefave, C. S. Redwood, E. Sparks, A. L. Cirino, S. Depalma, S. D. Colan, B. Funke, R. S. Zimmerman, and P. Robinson, "Familial dilated cardiomyopathy caused by an alpha-tropomyosin mutation : The distinctive natural history of sarcomeric dilated cardiomyopathy," *Journal of the American College of Cardiology*, vol. 55, no. 4, pp. 320–329, 2010.
- [9] H. D. Theiss, D. Robert, M. G. Engelmann, B. Andreas, S. Klaus, N. Michael, R. Bruno, S. Gerhard, and F. Wolfgang-M, "Circulation of cd34+ progenitor cell populations in patients with idiopathic dilated and ischaemic cardiomyopathy (dcm and icm)," *European Heart Journal*, vol. 28, no. 10, p. 1258, 2007.
- [10] X. Li, J. C. Lv, and Y. Zhang, "Manifold alignment based on sparse local structures of more corresponding pairs," in *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.
- [11] X. Li, J. Lv, and Z. Yi, "An efficient representation-based method for boundary point and outlier detection," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 1, pp. 51–62, 2018.
- [12] —, "Outlier detection using structural scores in a high-dimensional space," *IEEE transactions on cybernetics*, 2018.
- [13] X. Li, J. Lv, X. Wu, and X. Yu, "A semi-supervised manifold alignment algorithm and an evaluation method based on local structure preservation," *Neurocomputing*, vol. 224, pp. 195–203, 2017.
- [14] Y. Jiang, Z. Li, L. Zhang, and P. Sun, *An Improved SVM Classifier for Medical Image Classification*. Springer Berlin Heidelberg, 2007.
- [15] B. Li and Q. H. Meng, "Tumor ce image classification using svm-based feature selection," in *Ieee/rsj International Conference on Intelligent Robots and Systems*, 2010, pp. 1322–1327.
- [16] G. M. Foody and A. Mathur, "Toward intelligent training of supervised image classifications: directing training data acquisition for svm classification," *Remote Sensing of Environment*, vol. 93, no. 1, pp. 107–117, 2004.
- [17] D. S. Deshpande, A. M. Rajurkar, and R. M. Manthalkar, "Medical image analysis an attempt for mammogram classification using texture based association rule mining," in *Computer Vision, Pattern Recognition, Image Processing and Graphics*, 2014, pp. 1–5.
- [18] H. I. Kim and R. H. Park, "Residual lstm attention network for object tracking," *IEEE Signal Processing Letters*, vol. PP, no. 99, pp. 1–1, 2018.
- [19] Y. Gao, Y. Chen, J. Wang, and H. Lu, "Reading scene text with attention convolutional sequence modeling," 2017.
- [20] Y. Sun and R. Fisher, "Object-based visual attention for computer vision," *Artificial Intelligence*, vol. 146, no. 1, pp. 77–123, 2003.
- [21] F. Wang and D. M. J. Tax, "Survey on the attention based rnn model and its applications in computer vision," 2016.
- [22] F. Miau, C. S. Papageorgiou, and L. Itti, "Neuromorphic algorithms for computer vision and attention," in *International Symposium on Optical Science and Technology*, 2001.
- [23] M. M. Mahsuli and R. Safabakhsh, "English to persian transliteration using attention-based approach in deep learning," in *Electrical Engineering*, 2017, pp. 174–178.
- [24] H. Wang, S. Tang, Y. Zhang, T. Mei, Y. Zhuang, and F. Wu, "Learning deep contextual attention network for narrative photo stream captioning," in *Thematic Workshops of ACM Multimedia*, 2017, pp. 271–279.
- [25] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *Computer Vision and Pattern Recognition*, 2017, pp. 6450–6458.
- [26] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2016.
- [27] D. Balasubramanian, M. C. Krishna, and R. Murugesan, "Convolution-based interpolation kernels for reconstruction of high resolution emr images from low sampled k-space data," in *International Conference on Conference on Computational Intelligence and Multimedia Applications*, 2007, pp. 308–313.

Discovering Indicators for Classifying Wikipedia Articles in a Domain

A Case Study on Software Languages

Marcel Heinz¹

Ralf Lämmel¹

Mathieu Acher²

¹SoftLang Team, CS Faculty, University of Koblenz-Landau, Germany

²Univ Rennes, Inria, CNRS, IRISA, France

Abstract

Wikipedia is a rich source of information across many knowledge domains. Yet, recovering articles relevant to a specific domain is a difficult problem since such articles may be rare and tend to cover multiple topics. Furthermore, Wikipedia's categories provide an ambiguous classification of articles as they relate to all topics and thus are of limited use. In this paper, we develop a new methodology to isolate Wikipedia's articles that describe a specific topic within the scope of relevant categories; the methodology uses supervised machine learning to retrieve a decision tree classifier based on articles' features (URL patterns, summary text, infoboxes, links from list articles). In a case study, we retrieve 3000+ articles that describe software (computer) languages. Available fragments of ground truths serve as an essential part of the training set to detect relevant articles. The results of the classification are thoroughly evaluated through a survey, in which 31 domain experts participated.

1 INTRODUCTION

Wikipedia is a very large-scale, continuous community effort to collect and organize (informal) knowledge in almost all domains. In general, the quality of the articles and the mere principles of collection and organization challenge automated procedures in the quest of extracting and structuring Wikipedia knowledge [1–4]. Recovering articles describing topics in a certain domain is a reoccurring problem [5–7]. This paper presents a supervised machine learning approach for recovering Wikipedia articles relevant to an ontological class.

We conduct a case study on *software languages* defined as a set of digital artifacts, for which syntax, type system, semantics, and pragmatics can be (in)formally defined, documented, and implemented [8]. For the purpose of this research, we assume that Wikipedia's notion of "computer

language" – a notion that is used all across computer science and beyond – is essentially equivalent to the notion of "software language"; see also [9] for a discussion.

An important barrier is that, in a large number of cases, a *single Wikipedia article covers multiple subjects*. Looking at the very first sentences of the article about MATLAB (see Figure 1), several topics are mixed together (e.g., user interfaces, numerical computing, and software languages). The category graph is impacted as well (e.g., linear algebra and array-programming languages are both mentioned). There are also two infoboxes—one about MATLAB the *language*, the other about MATLAB the *software*.

Moreover, Wikipedia's category graph cannot be directly used for classification within the domain [10]; categories serve purposes other than classification, for example, collecting articles related to a common topic; see [https://en.wikipedia.org/wiki/Category:Java_\(programming_language\)](https://en.wikipedia.org/wiki/Category:Java_(programming_language)). Wikipedia's guidelines partly explain why multiple topics appear in an article: there should not be a new article, when the description benefits from explaining two or more subjects in the context of an existing article¹. Such rules and editing practices, though reasonable, complicate *classification*, i.e., automated identification of articles (instances) and underlying categories (classifiers). For some domains, like animals [5], categories that represent scientific classification are consistently used and provide a decision ground for still identifying relevant articles. In the domain of software languages such crucial features do not exist for articles or are used inconsistently. Alas, the recovery of relevant articles becomes looking for needles in a hay stack.

Significance of the multiple-subject problem. We initially explored articles that link to (subcategories of) the category 'Computer languages' and search for the top ten frequent nouns ('NN' tag recovered with part-of-speech tagger). We noticed that nouns from the music and astronomy domain are dominant. Fig. 2 reports the number of articles for the top ten most frequent nouns in the category

DOI reference number: 10.18293/SEKE2019-126

¹https://en.wikipedia.org/wiki/Wikipedia:Notability#Whether_to_create_standalone_pages

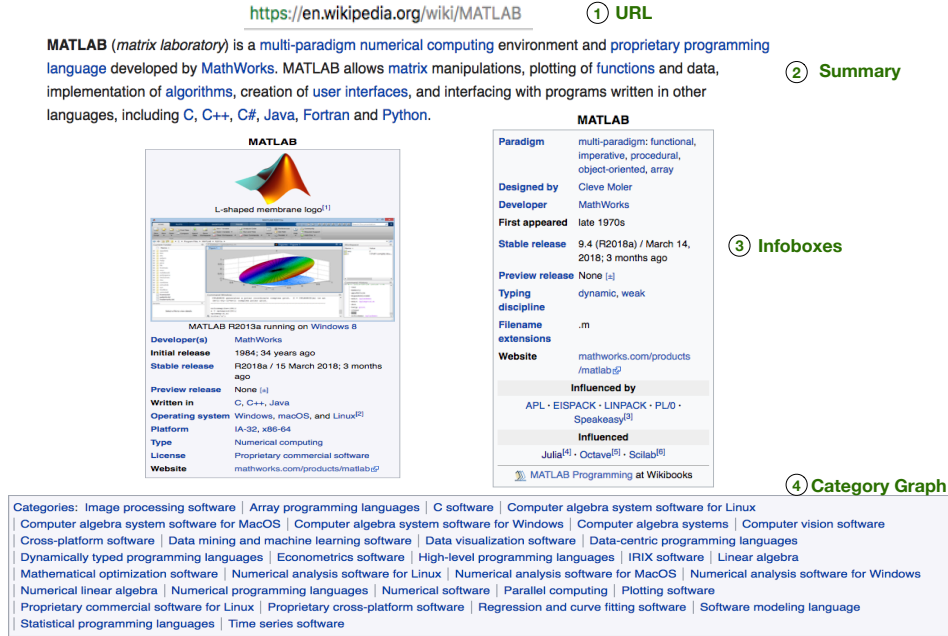


Figure 1. General structure of a Wikipedia article and some indicators.

tree of 'Computer languages' tree that we cut off at a depth of seven. Within the category tree, we observed subjects that belong to other domains, such as 'songs', 'stars', 'album' and 'music'. Except for the minority that describes domain specific languages in the respective domains, most of the article are obviously irrelevant to software languages. Fig. 2 also provides evidence that relatedness-based sub-categorization connect to irrelevant subjects.

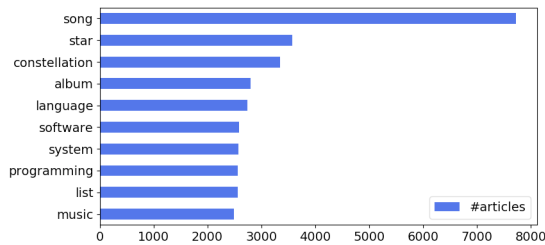


Figure 2. Top 10 most frequent nouns in articles below 'Formal languages' category.

Research question and contributions. Overall, our case study is an instance of the problem of extracting domain-specific knowledge from Wikipedia [7, 10, 11]: *How can we classify Wikipedia articles by their relevance to a given domain when relevant articles are rare and multiple main topics are covered by articles?* Our objective is essentially to detect members (here: Wikipedia articles) of a certain class (here: software languages).

We develop a seed-based learning methodology for identifying articles relevant to a domain-specific class while leveraging the available limited ground truth (based on

Github and TIOBE) and identifying indicators by inspecting a learnt decision tree that include URL patterns, summary text, infoboxes, list articles and category graph. We then present a case study which, in itself, results in the most comprehensive corpus on software languages available today. The results are evaluated by domain experts through a survey.

2 SEED-BASED LEARNING

We now detail how we exploit fragments of ground truths and instrument a decision tree classifier. The datasets including plotted decision trees are available online². Figure 3 provides an overview of our approach. i.) For training, we recover articles describing elements that appear in trustworthy external resources. ii.) Based on such seed, we define the scope in which we want to isolate relevant articles and label 4000 randomly sampled articles from this scope for additional training data. iii.) From the training data, we build a feature matrix with categorical values that state whether a structural feature, such as the programming language infobox template, is present. iv.) We configure a binary classifier that decides whether an article is relevant.

2.1 Seed Matching

As first step, we have identified and reused two trustworthy external data sets: both act as *seeds* for recognizing relevant articles. *GitHub* presents statistics on which languages

²<https://github.com/softlang/wikionto/>

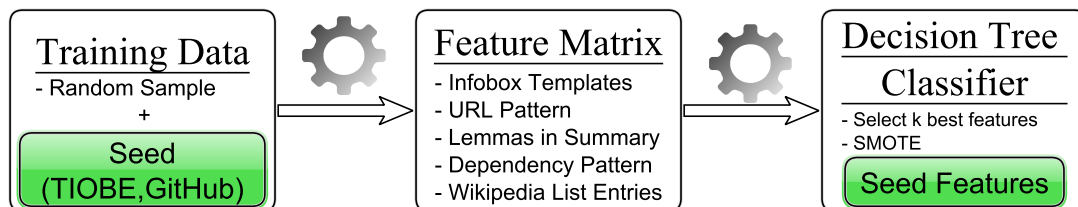


Figure 3. The ingredients of the approach. The seed in the training data influences decisions.

are used for any repository. The complete list of languages that are recognized can be extracted³. The *TIOBE* index presents statistics on how often software languages (mainly programming languages) are mentioned on the web. The list includes variations of names⁴.

Identifying each language on Wikipedia from the lists is challenging. We encountered favorable and problematic cases. To our favor, most names can be matched with article titles by leaving out annotations (e.g. '(programming language)') or by using Wikipedia redirects. However, manual reviewing remains crucial to make sure that languages are matched correctly, e.g., 'Red' is matched with 'Red (programming language)' and not with the color. In some cases, we succeeded by varying the writing style for the name, e.g., by unfolding acronyms, such as EBNF. In less favorable cases, we had to rely on Wikipedia's search engine to detect mentions in the text of existing articles. Only if the existing article describes the seed language as its main topic in the summary, we take it as the recalled article. To ensure availability of the necessary features for classification, we exclude stub articles from further analysis.

The resulting merged seed contains 327 *recalled seed languages* where 110 *overlap*. 158 seed languages are only casually mentioned in articles and 99 seed languages remain unrecognized.

2.2 Category Scope Exploration

We narrow the scope of our analysis to articles that are reachable by links to (subcategories of) chosen upper categories. Technically, we have manually identified common upper categories so that the seed is linked to them. At first, we hypothesized that all relevant languages link to 'Computer languages', but we found 'Augmented BackusNaur form' as a seed member that links to 'Formal languages', where 'Computer languages' is a subcategory. Moreover, for seed articles such as 'CSV', we noticed the upper category 'Computer file formats' that is disconnected from 'Formal languages'. Such formats are categorized in a different manner, but they do conform to the definition of software language. Our experiments show that all seed articles are

Table 1. Number of seed articles per depth.

	0	1	2	3	4	5	6-8
Formal languages	2	6	85	136	70	16	7
Computer file formats	8	22	11	7	0	0	3

linked to (subcategories of) two disconnected roots in the category tree at a maximum depth of eight. Table 1 illustrates at what depth from the chosen root categories seed articles exist.

2.3 Feature Extraction

Next, we describe which features we extract from the content types discussed in Section 1. To counter lack of decisive features and to reduce dimensionality, we exclude stub articles and only consider features that appear in at least ten articles. For now, we also exclude articles at a distance to the upper categories that is higher than 8. As a resulting dataset, the extraction returns an article-feature matrix with $104,186 \times 46,173$ entries with label '1' for present and '0' otherwise. Then, the set of 4000 randomly sampled articles and the seed form the training set. To avoid memory issues and enable loading the whole article-feature matrix, we use sparse matrices.

In the evaluation, we discuss that irrelevant articles in the training set are actually members of many other domain classes and have many different features. Hence, the resulting decision tree decides by features present in seed articles. In fact, GitHub and TIOBE seeds have a more general interest: **Seed articles provide representative features for recognizing relevant articles.** We show which features of seed articles provide a representative positive indication and hence may be found in a fit decision tree.

Infobox Templates. In the scope, there exist 864 distinct infobox template names. 263 seed articles use an infobox template. We found the templates on 'programming language' (215), 'file format' (32), 'technology standard' (3) and 'software license' (1). Especially, the templates 'programming language' and 'file format' provide a strong positive indication, but they do not exist for every relevant article.

URL Pattern. We extract all words separately in braces from every article's title as they provide a semantic anno-

³<https://github.com/github/linguist/blob/master/lib/linguist/languages.yml>

⁴<https://www.tiobe.com/tiobe-index/programming-languages-definition/#instances>

tation for disambiguation. Only around a third of the seed articles has such an annotation. The words in braces appearing more than once with their frequency are: 'language' (125), 'programming' (114), 'software' (6), 'stylesheet' (3) and 'markup' (2). 'programming' always appears together with 'language'.

Lemmas in Summary. From the summary, we extract all words, filter them by a stop word list and apply lemmatization on the remainder. We recovered 5449 lemmas from seed articles while 457, 164 distinct lemmas can be recovered from all articles. We enumerate the top five lemmas from the seed: 'language' (301), 'programming' (253), 'use' (216), 'develop' (120), 'design' (117). Lemmas are often used for topic models [12]. They are essential to distinguish the domain of software languages and co-occurring topics in articles from other unrelated domains such as natural languages.

Dependency Pattern. Following [10], we identified multiple kinds of relationships that can be extracted from the first sentence. Hypernyms provide a reliable feature as most articles on Wikipedia begin with a sentence containing 'is a'. Based on the dependency graph, we extract the hypernym [13], if the sentence does not start with 'A'. This allows us to ignore subset relationships as in 'A programming language is a formal language[...]'. To reach higher coverage, we infer instantiation from relationships such as part-of as in 'Perl 6 (also known as Raku[5]) is a member of the Perl family of programming languages', where 'languages' is the extracted hypernym. In the seed, we found 'language' (235), 'format' (16) and 'dialect' (10) as the most frequent hypernyms. The hypernym 'language' and 'dialect' are also recovered when analyzing articles on natural languages.

Wikipedia List Entries. As an internal resource, we consider Wikipedia lists of things [14]. Such lists collect names of entities of a certain kind and optionally provide additional information, e.g., 'List of dog breeds'. First, we recover the list of lists by searching for 'list of' in the title of articles in the scope. Binary Features state in which lists an article is linked. In the seed, we found 267 articles linked in such lists, such as the 'List of programming languages'.

3 EVALUATION

We investigate the results from evaluating a classifier based on labels by experts (RQ1). Since we chose to use a decision tree classifier, we can present technical insights on how decisions are made and explain the effects of using a representative seed (RQ2). We conclude with presenting how many software language articles and categories are estimated in the scope (RQ3).

3.1 Precision & Recall (RQ1)

We provide qualitative insights on **How well does the classifier perform beyond the seed?**

To gain an evaluation set that is as objective as possible, we conducted a survey in which 31 domain experts from our research groups and external collaborators participated. In each question in the survey, participants decided whether a presented article explicitly describes a software language as one primary topic. We received 990 articles labelled by at least two experts. In order to gain additional insights on problems with decision making, we emphasized the possibility of commenting on each question. For 43 articles, experts did not agree. The articles present border cases, for which experts are not sure. For example, articles do not directly refer to logic formulae as a software language, but such formulae are often digitally encoded, follow a syntax, well-formedness rules (thus, a type system) and have semantics and pragmatics. For the future of SLEBOK [8], such border cases can be used to further investigate on the borders of the term software language. A refinement of definitions would then again lead to better expert labels to, for now, problematic articles. Such refinement hopefully leads to a better categorization on Wikipedia itself.

As long as experts cannot reliably decide for an article whether a software language is described, a machine cannot as well. As a consequence, we exclude these articles. Since only ~4% is excluded, the threat to validity is reasonably low. We took the remaining expert labels as the evaluation set and explored several configurations with a k-best feature selection and synthetic oversampling with SMOTE [15].

With $k = 23$, the learned classifier performs with an *f1-score* of 0.7, *balanced accuracy* of 0.9, *recall* of 0.81 and *specificity* of 0.99.

3.2 Indicator Discovery (RQ2)

For imbalanced datasets, a classifier usually overfits towards the major class unless countermeasures are taken [15]. This problem is countered in two ways. i.) By adding the seed as an addition to the random sample to our training set, the rate of relevant articles in it is not representative which can be seen as a form of manual oversampling. ii.) Actually, we isolate articles relevant to a single class from articles relevant to many other classes (e.g., songs, stars, software). Feature selection based on *Gini index* or *entropy* pick features according to their score in discriminating classes, in our case relevant versus not relevant. Consequently, the features frequently appearing in seed articles are preferred, since the irrelevant articles have a huge amount of different features and thus single features recall a lesser percentage. At last, synthetic oversampling with SMOTE further improves results. This peculiarity mo-

Table 2. How many articles and categories exist in *total*, identified by the *seed*, and classified as *relevant*.

	Articles			Categories		
	<i>Total</i>	<i>Seed</i>	<i>Relevant</i>	<i>Total</i>	<i>Seed</i>	<i>Relevant</i>
Formal Languages	101641	301	2897	21822	353	1339
Computer File Formats	6116	46	745	235	18	79

tivates further inspection of the learned decision tree as an answer to **what features indicate articles on software languages**.

The templates ‘programming language’ and ‘file format’ provide a strong positive indication, but they do not exist for every relevant article. While the hypernym ‘language’ also provides strong indication, it cannot be used alone. Otherwise, natural language related articles are confused to be relevant. Lemmas, lists and URL pattern help in discriminating relevant articles from articles with overlapping features. Below, we list the most important indicators in the decision tree categorized by their source (see Section 2.3).

Infobox Templates: file format, programming language; **URL Pattern:** programming, language; **Lemmas:** syntax, code, programming, compiler, design, general-purpose, language, support, compile, object-oriented, use; **Dependency Pattern:** language; **Wikipedia List Entries:** List of programming languages, List of programming languages by type, List of file formats, List of C-family programming languages, List of object-oriented programming languages.

3.3 Article and Category Relevance (RQ3)

We give quantitative insights as an answer to the question: **How many articles and categories remain relevant for the domain of software languages?** Table 2 summarizes the degree of the reduction per root category based on the configuration from Section 3.1. **Based on the predicted reduction, we find 2797 more articles in the scope than there are already in the seed.** That is, we significantly augment the identification of software languages.

For the root category ‘Computer file formats’, the classifier predicts a lot of irrelevant articles. When inspecting infobox templates used in these articles, we observe a topic mix with, for example, software, websites and companies. Here, a threat remains that the articles do not directly address formats in several articles in a recognizable way. In comparison to ‘Formal languages’, a higher percentage of categories is estimated as relevant. Accordingly, we observe that the subcategory tree is more consistently maintained.

Inspecting single categories backs up subjective hypotheses based on manual inspection of the usefulness of specific categories. For example, out of 22546 articles that are (transitive) members ‘Statistical data types’, only 52 are classified as relevant. The number of articles indicated to be

relevant in a category becomes a more objective estimation for its usefulness.

Table 2 provides a summary when inspecting all categories based on the assumption that the number of relevant articles hints at the relevance of categories. Out of 21822 categories below ‘Formal languages’, 353 categories contain seed members and 1339 categories remain relevant. **Only 6% of transitive subcategories under ‘Formal languages’ are estimated to be useful within the scope.**

4 RELATED WORK

Previous attempt: In [16], the category graph is pruned by manually excluding categories not serving for classification of software languages in a common sense, subject to a small suite of criteria. In contrast to our work, the approach relies on manual selection of categories instead of automatically classifying articles by combining several indicators.

Domain ontology: Wikipedia is a frequent target for knowledge discovery. *To the best of our knowledge, no approach tries to detect articles describing a class from an inconsistently maintained domain, where relevant articles are rare and require to combine multiple indicators.* For deriving domain ontologies, various distinct approaches can be found. The approaches range from supporting manual crafting [17, 18] to unsupervised crafting from text [12]. Wang et al [5] extract a domain ontology for animals from Wikipedia based on the category graph, article graph and section structure in articles. For the animals domain, most pages are maintained well and describe exactly one concept in a consistent order. Mirylenka et al [10] extract subset, membership, part-of, sub-topic and other relationships by analyzing the varying nature of subcategorization. Dong et al [6] learn subsumptions from articles describing domain concepts based on Hearst pattern. In a related approach [7], the concept set is learned by matching articles with Stackoverflow tags. Based on the article- and category graph, the concept set is expanded and further relationships are learned. Related works discover knowledge from different structural features, such as the title [10, 19], text [13, 20], an article’s section structure [5], links to other articles [7, 20], infoboxes [3], lists [14], etc.

External knowledge: The usefulness of WordNet varies depending on the coverage of terms in a domain. While WordNet is a known assistant in more common knowl-

edge domains, low coverage is reported in more specific domains [2, 5]. Our experience confirms it for software languages. Various knowledge graphs exist. *DBpedia Live* [21] mirrors Wikipedia while refining it with more ontological knowledge. YAGO [4] is a knowledge graph derived from Wikipedia, WordNet and Geonames that explicitly focuses only on structured knowledge aspect that is then again linked by Dbpedia as well. Wikidata [18] is widely manually crafted and tries to reach a clean knowledge graph from scratch. A type encompassing the term software language is not maintained by any of the mentioned resources.

5 CONCLUSION

In a domain where only a small set of positively labeled articles can be used as a ground truth, we did find strong indicators for classifying 3000+ articles as relevant for software languages. We showed that a learned decision tree classifier provides reasonably high recall and low false-positive-rate and allows one to inspect isolated articles inside Wikipedia. While learned random forests might provide higher accuracy, we are specifically interested in higher interpretability of decision trees. 31 domain experts thoroughly evaluated our classifier by labelling over 990 Wikipedia articles.

A natural follow up of this work is a collaborative engagement with experts to further extract and improve fine-grained classification knowledge into a high-quality domain ontology that will help students and professional developers to better understand software languages. We are in the process of repeating the experiments for the domain of software (in a broad sense). The general problem is the same: We are confident that our learning methodology can identify the body of domain knowledge within Wikipedia. There are also specific challenges ahead due to particularities of the domains (for example, 'Computing platforms' as a category is disconnected from 'Software').

Acknowledgements. This research was partially funded by the ANR-17-CE25-0010-01 VaryVary project.

References

- [1] B. Stvilia, M. B. Twidale, L. C. Smith, and L. Gasser, "Information quality work organization in wikipedia," *JASIST*, vol. 59, no. 6, pp. 983–1001, 2008.
- [2] Q. X. Do and D. Roth, "Exploiting the wikipedia structure in local and global classification of taxonomic relations," *Natural Language Engineering*, vol. 18, no. 2, pp. 235–262, 2012.
- [3] F. Wu and D. S. Weld, "Automatically refining the wikipedia infobox ontology," in *Proc. WWW*, 2008, pp. 635–644.
- [4] T. Rebele, F. M. Suchanek, J. Hoffart, J. Biega, E. Kuzey, and G. Weikum, "YAGO: A multilingual knowledge base from wikipedia, wordnet, and geonames," in *Proc. ISWC 2016*, 2016, pp. 177–185.
- [5] H. Wang, X. Jiang, L. Chia, and A. Tan, "Wikipedia2Onto. Building Concept Ontology Automatically, Experimenting with Web Image Retrieval," *Informatica (Slovenia)*, vol. 34, no. 3, pp. 297–306, 2010.
- [6] X. Dong, K. Chen, J. Zhu, and B. Shen, "Learning to discover subsumptions between software engineering concepts in wikipedia," in *Proc. SEKE 2016*, 2016, pp. 147–152.
- [7] K. Chen, X. Dong, J. Zhu, and B. Shen, "Building a domain knowledge base from wikipedia: a semi-supervised approach," in *Proc. SEKE 2016*, 2016, pp. 191–196.
- [8] B. Combemale, R. Lämmel, and E. V. Wyk, "SLEBOK: The Software Language Engineering Body of Knowledge (DS 17342)," *Dagstuhl Reports*, vol. 7, no. 8, pp. 45–54, 2018.
- [9] J. Favre, D. Gasevic, R. Lämmel, and A. Winter, "Guest Editors' Introduction to the Special Section on Software Language Engineering," *IEEE Trans. Software Eng.*, vol. 35, no. 6, pp. 737–741, 2009.
- [10] D. Mirylenka, A. Passerini, and L. Serafini, "Bootstrapping domain ontologies from wikipedia: A uniform approach," in *Proc. IJCAI 2015*, 2015, pp. 1464–1470.
- [11] M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López, "The neon methodology framework: A scenario-based methodology for ontology development," *Applied Ontology*, vol. 10, no. 2, pp. 107–145, 2015.
- [12] S. Sarencheh and A. Schiffauerova, "Automatic algorithm for extracting an ontology for a specific domain name," in *Proc. KEOD*, 2017, pp. 49–56.
- [13] T. Flati, D. Vannella, T. Pasini, and R. Navigli, "Two Is Bigger (and Better) Than One: the Wikipedia Bitaxonomy Project," in *Proc. ACL*, 2014, pp. 945–955.
- [14] P. Kuhn, S. Mischkewitz, N. Ring, and F. Windheuser, "Type inference on wikipedia list pages," in *46. Jahrestagung der Gesellschaft für Informatik*, 2016, pp. 2101–2111.
- [15] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.
- [16] R. Lämmel, D. Mosen, and A. Varanovich, "Method and Tool Support for Classifying Software Languages with Wikipedia," in *Proc. SLE 2013*, 2013, pp. 249–259.
- [17] M. Völkel, M. Krötzsch, D. Vrandečić, H. Haller, and R. Studer, "Semantic wikipedia," in *Proc. WWW*, 2006.
- [18] D. Vrandečić, "Wikidata: a new platform for collaborative data collection," in *Proc. WWW*, 2012, pp. 1063–1064.
- [19] R. Zarrad, N. Doggaz, and E. Zagrouba, "Title-based approach to relation discovery from wikipedia," in *Proc. KEOD*, 2013, pp. 70–80.
- [20] V. Presutti, S. Consoli, A. G. Nuzzolese, D. R. Recupero, A. Gangemi, I. Bannour, and H. Zargayouna, "Uncovering the Semantics of Wikipedia Pagelinks," in *Proc. EKAW 2014*, 2014, pp. 413–428.
- [21] M. Morsey, J. Lehmann, S. Auer, C. Stadler, and S. Hellmann, "Dbpedia and the live extraction of structured data from wikipedia," *Program*, vol. 46, no. 2, pp. 157–181, 2012.

Model View Controller in iOS mobile applications development

Dragoş Dobrean and Laura Dioşan

Faculty of Mathematics and Computer Science, Babes Bolyai University

Cluj Napoca, Romania

{dobrean,lauras}@cs.ubbcluj.ro

Abstract—Due to the increased number of mobile applications and their popularity, many software developers have begun to focus on mobile platforms. While this focus has positive effects (e.g. a larger developer community, new open source projects, new tools), it also has a down side. With the migration of developers from different software development areas, where they have used other programming paradigms or architectural approaches, the topic of software architecture on mobile platforms become more trending and hype in the mobile development communities. Even though several new architectural solution were proposed for solving some of the issues which arise from using the classical architectural patterns popularised by the creators of the mobile platforms, we want to emphasise the principles of software architecture in mobile computing, why they have to be respected and how their adoption impacts the development process. Therefore, this paper focuses on showing that the Model View Controller (MVC) — one of the most common classical architectural patterns — can be used successfully for building mobile applications and the problems which might arise are by products of the wrong usage of the pattern rather than pattern issues. We show that by analysing the most common architectural misuses of the MVC pattern in both open-source and private projects and offers solutions to those problems.

Index Terms—Apple's Model View Controller (MVC), Mobile Software Architecture, Architectural Smell, iOS.

I. INTRODUCTION

Mobile applications have become an important part in the life of the modern man. The involved devices have become indispensable companions in our lives. We use them for social interactions as well as for business activities, for increasing our productivity or for self improving and entertainment. According to GSMA Intelligence two thirds of the world are connected by mobile devices [1]. In order to be able to sustain this high rate of adoption and popularity of the mobile applications, many developers migrated from building other types of software to building mobile applications.

This blend of domains has brought many new trends to the mobile platforms (for instance, functional programming which has become used in large mobile projects [2], [3]). However, this union of domains has also had downsides: new or inexperienced developers of those who have migrated from other platforms did not understand fully how all the mobile development concepts work, how are they supposed to be used, and there were not many places in which they could learn how

to properly use those. Mobile software architecture has become greatly affected by this phenomena [4]–[6].

It is an established fact that a good software architecture and design could increase the system quality: performance, evolvability, maintainability and reliability [7]. When weak design decisions affect the software properties, the architecture is often subject to various problems (code smells, design smells or architectural smells). Related work regarding software architectural smells can be found in [8] where the authors showcase 11 architectural smells grouped in 4 different categories (Interface-Based Smells, Change-Based Smells, Concern-Based Smells and Dependency-Based Smells). All of issues can also apply to mobile platforms software applications. In [9] and [10] 4 of the architectural smells are analysed in depth and shown on 2 industrial software systems. Other work has been done in [11] where the authors have analysed the architectural erosion of Open-Source Software projects.

Although the drawbacks of architectural erosion have been already recognized [9], [10], [12], [13], the authors of these studies have focused on classical systems, rather than mobile ones: Velasco et al. [14] presented several architectural smells that are relevant to MVC, while Aniche et al. [15] identified six classes in the context of web applications constructed on MVC pattern.

Our initial effort drives to identify the possible MVC problems in the context of mobile applications, to characterise them and to describe how they can be fixed. In this paper, we aim to answer the following research questions:

- RQ1: What are the problems of MVC in the context of mobile applications?
- RQ2: Is there a classification for the MVC architectural problems?
- RQ3: How can the problems be fixed?

To the best of our knowledge, an MVC analysis in the iOS context was not been performed until now. Previous works [14], [15] are focused on web or classic flavours of MVC and on the problems caused only by the violation of constraints which MVC style defines between its components or layers. In the mobile context, the weaknesses of Android's design have been analysed and a passive flavour of MVC was proposed in [16].

The rest of the paper will focus on the iOS platform on its architectural pattern encouraged by Apple, Model View Controller (MVC) [17]. Architectural issues explained in this

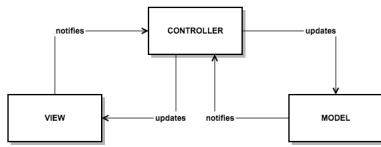


Fig. 1. Apple's Model-View-Controller architectural overview

paper also apply to other platforms which use MVC for building applications (macOS, Windows, etc.), while the naming conventions might be different the concepts and issues are the same. MVC was chosen as is one of the most know presentation architectural pattern while being present on all the mobile applications platforms in various flavours. In addition, MVC is highly versatile, having different flavours as Model-View-View-Model (MVVM) [18], Model View Presenter (MVP) [19], etc.

The following section talks about MVC and Apple's flavour of it. Section

II. APPLE'S MVC

Model View Controller is one of the most widespread presentational architectural patterns, being used to create desktop, mobile and web applications [20]–[22]. The purpose of MVC is to provide a simple separation of concerns for an application that embeds a user interaction component. One of the most important aspects of this pattern is that the application should work, and fulfil all its requirements even if we remove the View and Controller layers. The data manipulation and the business logic should reside the Model and it should not be affected in any way by those other layers.

Apple's version of MVC [17] is different from the conceptual, generic one [23]. The classic MVC [23] was coined when there did not exist the concept of mobile applications; in order to compensate for this fact, Apple promoted a flavour of MVC, which is better suited for mobile and desktop applications.

It is composed from the same three layers (Model, View, Controllers) and the only thing changed is their way of interaction and the data flow. The layers are more decoupled and it does not rely so heavily on the Observer/Delegate pattern; it is still being used, but it is not as used as intensively as on the classical pattern. In Apple's flavour, the Observer/Delegate pattern is more aimed to provide callbacks from one layer to another than to observe Model layer properties.

The accent in the Apple's flavour of MVC is on the Controller, as can be seen in Fig.

This emphasis shift can also be seen in the way they have named their framework components. At the centre of every iOS application, we find the View Controllers which act as bridges between the data of the application (Model) and the user interfaces (View).

In Fig.

Advantages This flavour of MVC simplifies (from the classic MVC) the data flow between the layers, making the data flow clearer. In addition, it also reduces the coupling

between the layers; the link between the Model and The View (from the classic MVC) no longer exists, making the components more isolated.

Disadvantages The Controller layer becomes the central piece of the architecture; it needs to ensure the proper communication between the Model and the View layer and vice-versa. This task makes the Controller layer to grow to be quite complex, which can lead to architectural issues (massive view controllers) as well as OOP issues (violation of single responsibility principle).

III. ANALYSIS

As instances of poor design decision, the architectural smells originate in the improper use of a design solution or of software architecture-level abstractions. In what follows we attempt to facilitate the identification of such problems in the context of mobile applications involving the MVC pattern. We provide a short description of each problem and its causes (trying to answer RQ1), the architectural smell's class it belongs to (as answer to RQ2) and one or more possible solutions (answer to RQ3).

In order to be able to answer those questions we needed to inspect different sized codebases, we have chosen 5 codebases, with different variations of MVC that added architectural layers from: MVVM [18], MVP [19] VIPER [24].

- Wikipedia – education and information app [25]
- Firefox – a mobile web-browser [26]
- Trust – cryptocurrency wallet [27]
- E-Commerce — private
- Game – private

TABLE I
CODEBASES SIZE

Application	Blank	Comment	Code	Source
Firefox	23392	18648	100111	open-source
Wikipedia	6933	1473	35640	open-source
Trust	4772	3809	23919	open-source
E-Commerce	7861	3169	20525	private
Game	839	331	2113	private

As can be seen in Table

A. Complexity

1) *Issues:* Mobile applications have grown to be complex software systems where they do much more than just fetching some data from a web service and displaying them on the screen. While for a simple application the MVC pattern would be sufficient, if we add extra complexity, such as working with databases, caching, virtual reality, audio, photo or video manipulation, we might ran into some issues.

Although using another presentational pattern (e.g. MVVM [18] or MVP [19]) might solve some of the problems, if the application becomes more complex, even these patterns will not be sufficient to maintain it flexible to change, testable and easy to be understood and worked on.

2) *Solutions:* MVC is an architecture to be used on small-medium sized applications. If we talk about a complex application then an architectural approach needs to be designed in order to fulfil its use-case. MVC provides the basis for this new architecture and its three degree of separation should definitely be implemented. But, in addition and in order to make the codebase maintainable, we might introduce additional layers, such as Presenters from MVP, View - Models from MVVM or Routing objects from VIPER [19], [24], [28], [29].

3) *Findings:* Trust, E-Commerce and the Game had the clearest defined architecture. Analysing the codebase is easy to get a grasp on how the app works and the codebases were consistent in both naming and design pattern used. Firefox, being the largest codebase and given its functionality, is a more complex one and it is the hardest from the codebases to understand and uses multiple layers. Wikipedia relies on multiple open source libraries and internal UI libraries which introduce extra obfuscation and makes the codebase harder to comprehend.

B. Misunderstandings

1) *Issues:* Another common problem we encounter when talking about MVC is that people usually have different concepts of what MVC is and how should it be used. People coming from ASP.net MVC [30] development might have a total different idea of how the MVC components should communicate from those who are developing mobile applications. As we have already seen, different companies have different definitions for what MVC is and how it should be used with their frameworks. This problem has a real impact when developers migrate from one platform to another and try to use the same MVC definitions on a new framework.

Almost all people with some knowledge about the MVC agree that the Model should contain data useful for application [31], the View is responsible with presenting the data to the user and the Controller layer acts as a mediator of some sort between the other two layers. Those are very vague definitions of MVC and 2 people can disagree on what should be put in the Model and what should reside in the View or Controller layer.

Developers of MVC frameworks usually give their definition of what MVC meant for them or how should this be used; however, those definitions usually are vague as well. The reason of this ambiguity can be rooted in the generality of the frameworks, which should adapt to many types of applications. A constrained architecture, which would fulfil all potential use-cases is not feasible and it is considered redundant for applications that do not need a high level of architectural complexity.

2) *Solutions:* The first step into ensuring that an architectural pattern is correctly implemented is to have clear definitions of its elements and everyone involved in the project to be well aware of. In order for this to happen, it is important that the lead developer or the system architect to understand clearly the scope of the product and to be able to draw architectural guidelines which would fit the project.

3) *Findings:* From the analysed projects, Trust, Firefox and the E-Commerce app were the ones in which there was a clear defined architecture which could be inferred from the codebase and it was consistently used. The Game app was the smallest and its architecture didn't require any specific guidelines given its complexity. In the case of Wikipedia, the code was not consistent, and the guidelines were not clear.

C. Model

1) *Issues:* Most of the problems which appear in this layer are design pattern issues; the usage of too many singleton objects and the violation of SOLID principles [32] are the root cause of the problems which can appear at this level. The result will be a damaged architecture at a micro level — high coupling between items in the same layer.

Among the common mistakes in the model are the fact that objects which interact with a database or a web backend service have reference to the ones using those (usually View Controllers). These references can create retain cycles — Dependency-Based Smells [8] — and also impact the MVC architecture by making the Model layer have knowledge about the Controller layer.

The problems which appear at this level are usually from the Interface-Based Smells category as defined in [8], [11]. Is not uncommon to find Ambiguous Interfaces or Concern Overload where a component performs a large amount of tasks and have a scarce number of interfaces. For instance, the objects which communicate with the backend for fetching data are most of the time responsible for creating the connection, converting the input parameters to what types of information does the backend service expect, parsing of the response it receives and mapping it to a codebase defined entity.

2) *Solutions:* This issue can be solved using the Observer/Delegate pattern, where the Model layer provides callbacks for its events and the Controller layer takes various actions based on those events.

3) *Findings:* All the analysed codebases presented issue on the Model layer as each one of them has wrong, direct dependencies between the Model layer and the View or Controller layer. The most problematic ones were the E-commerce and Firefox. In all the codebases excepting the Game, we find the Concern Overload and the Ambiguous Interfaces smells [8], [11].

D. View

1) *Issues:* In large projects, which do not have major architectural issues, the Controller objects configure the Views by directly passing the Model item as an argument. This common practice creates a dependency between the View and the Model, which is not presented in the Apple's way of defining the MVC (Fig.

An example would be a list of new movies in a booking application: the cell that is responsible for displaying a new movie will usually receive from the Model, a Movie entity, which contains much more data then what is needed to be displayed (the ID from the database, a list of actors, number

of people who already booked it, etc.). This is an overlooked issue with MVC and usually the developers accept it, even if this is an architectural mistake nevertheless.

The mentioned problem belongs to Co-change Coupling smells [11], an architectural issue which occurs frequently at this level. The coupling is predominantly done between the View and the Model layer. However, this can also appear in the View and Controller layer.

2) *Solutions*: In order to overcome this difficulty, new objects can be defined for keeping the configuration of the view (when the view needs a lot of configuration information from the model), or this information can be passed as parameters to the view using primitive types.

The new defined items for the configuration of the View rise another problem: where should those items reside? They know nothing about the View so is not in the View layer; however, they are only used and have meaning in a context in which those Views exist. An approach to solve this problem would be to treat these items as belonging to the Model as they handle the business logic display part, they can be seen as mappers between entities and views or data transfer objects.

Another approach, if the developed application needs this kind of complexity, is to use another architectural pattern namely MVP [31], which inherits from MVC; basically, it is a variation of MVC, where there is a new layer for configuration objects, called Presenter. The Presenter, however, is a more specialised object: besides configuration information, it also contains information regarding the state of the View (selected, unselected, highlighted, whether or not some of the fields should be pre-filled etc.).

3) *Findings*: All the analysed codebases have shown Co-change Coupling smells, the most severe ones was the E-commerce one. The open-source apps also exhibited this issue, however at a much lower degree.

E. Coordinating Controllers

1) *Issues*: Just like in the case of Apple's MVC, there are different flavours of MVC where there is a combination of roles (View and Controller) into a single entity called View Controller. This entity owns the View and it responds to its events. The View Controller is responsible for responding to input received from the View and for displaying and moving those Views on the screen. Those kind of controlling objects usually derive from a superclass. For instance, on the iOS SDK, the superclass is `UIViewController`, on Android we have the `Activity` superclass.

There are cases where the complexity of the application requires another kind of controller objects — Coordinating Controllers. Coordinating type of controllers are simple objects that manage the application; they usually decide when a certain action should happen and keep track of the state of the application [33]. Those kinds of objects are responsible for deciding on what state (flow) of the application to go next (when a certain event occurred) based on the current state, for setting up the initial state and managing the lifecycle of contained objects.

By flow and state we mean what use case scenario is presented on the screen at a certain moment in time; flow is a broad term and in the context of this paper we are using it to describe a use case (e.g. sign up), if we were to have a higher granularity, the flow can be split in multiple sub-flows (e.g. the forgot password of the sign up flow).

Therefore, the Controller layer can be split into two categories: View Controllers and Coordinating Controllers. The View Controller objects have come to be generally accepted as the Controller objects by most of the practitioners in this field. However, this is not always the case and there is an important distinction between Coordinating Controller objects and View Controllers [17].

Unfortunately, this degree of separation in the context of Controller layer is not so well understood on the iOS platform. All these concepts appear in AppKit development scene (desktop application for macOS), as this platform is older and more evolved. Usually this form of separation within the Controller layer is not needed, as the applications are not complex enough to justify it. The problems start arising when the application becomes complex and the people working have a lack of architectural knowledge on how to scale it or the architectural state in which they need to arrive is unknown or insufficiently defined.

Frequently, the responsibilities of Coordinating Controllers get stuffed in the View Controller objects increasing their complexity and changing their purpose as now, they also have to take care of knowing the state of the application and correctly transitioning between the states in every possible configuration. By taking this responsibility in other custom objects (Coordinating Controllers), the View Controller object become slimmer and they are no longer depending on each-other.

This practice is fairly popular or familiar and is usually implemented in applications where a clear architectural guideline is not defined or not sufficiently described and defined for all the potential corner cases. A lighter common version of this coordination is to have an object which all the View Controllers inherit from, where all the common navigation flows are stacked in.

This sort of behaviour (merging responsibilities) is common for small applications where the UI is quite simple (one-three screens), where the extra Coordinating Controller objects would not provide real value, or for beginner developers. Most of the applications which are fairly complex have multiple flows (sign up, sing in, browse items, add to cart, checkout, previous orders, feedback, settings, profile, etc.). These applications are the ones which suffer massively from the lack of coordinating layer as their View Controller objects become bloated with navigation and configuration logic. This kind of complexity creates architectural issues especially when the application needs to be changed because many components fulfil the same functionality, for instance the correct navigation from one screen to another (Scattered parasitic functionality [8]).

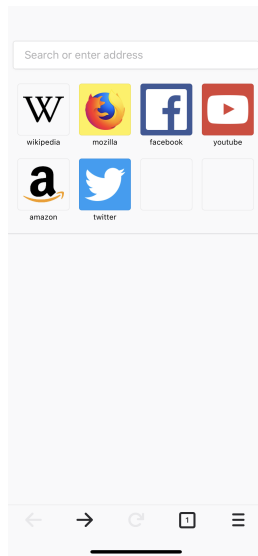


Fig. 2. Firefox iOS application screenshot [26]

2) *Solution*: The solution to this problem (complex application with multiple use-cases and flows) is to have multiple Coordinating Controller objects for every flow of the application or for every sub-flow of the application; each of these flows will have a single, well defined use case (login in the user, uploading a picture, making a payment, etc.). All the application's flows and navigation will be then expressed via those building blocks (Coordinating Controller objects for certain flows or sub-flows). By using this approach, we reduce the complexity of View Controller objects. They become concerned only with displaying the data and mediating between View and Model layers. All the navigation and configuration logic now resides in the Coordinating Controller objects. In addition, we can easily change the flows of the application even at runtime, we can Unit Test the navigation from one screen to another and the correct configuration of the View Controller objects, which, in the case of massive view controllers, is rather hard.

3) *Findings*: Firefox, Trust and the E-commerce apps were the ones in which the Coordinating Controllers were correctly used. Wikipedia was the worst analysed app from the Coordinating controllers point of view, the Scattered parasitic functionality [8] is predominantly present in the codebase.

F. View Controllers

1) *Issues*: Another issue which is overlooked is that developers usually use one view controller per screen (the UI elements shown on the full size of the screen). While this is the right approach for simple screens such as a "Terms and conditions" screens or even a "Login" screen, if we talk about complex UI interfaces (e.g. the browse screen from Firefox) this is totally wrong.

The browse screen from the Firefox application (see Fig.

A large amount of the problems which we encounter in the MVC approach on iOS deals with the View Controller.

In fact, it has access to both View and Model layers and acts like a binder between them; an example would be if the data obtained from the Model is not well formatted, the View Controller will format it for the View and this is not clearly its responsibility. A View Controller should only be concerned with presentational aspects of a certain part of the application and for handling the user input received from the View. Obviously problems can appear at other levels, as well on Coordinating Controllers or Model level, but these usually, like in the case of View Controllers, have the root cause the low granularity of the architectural components and can be solved by increasing the granularity of the elements (splitting a certain item in multiple others and use the Composition design pattern).

In addition, the View Controller objects are also bloated with handling View logic and states. By view logic and state we mean keeping the internal state of the view which cannot be inherited from the Model. For instance, knowing which items were selected on the screen, what slider is enabled etc., before applying these changes to the Model. This sort of logic should be implemented in custom objects; most of the times the MVP pattern is used for solving this issue, but as a workaround in MVC, these can reside in subclasses of View components or in custom objects defined in the Model layer.

At this level, the major architectural smell is Concern Overloading [11], as like previously shown, the View Controller object become bloated with an excessive amount of responsibilities.

2) *Solutions*: The solution to this problem is to use multiple view controllers for the UI elements; for instance, we could have a View Controller object responsible for the turn by turn navigation advices, we could have another one for the map and so on. Furthermore, if these elements are complex by their own, they could be split further in more View Controller objects which should respect the single responsibility principle. By using this approach we would obtain view controllers that respect the single principle responsibility ensures a good separation of concerns, they contain less code, and they become testable.

As in the case of Coordinating Controllers where we could have Coordinating Controller objects, which depend on other Coordinating Controller objects we can apply the same logic to creating user interfaces and using multiple child View Controller objects to construct a single screen of the application. By using this approach, each View Controller object will have single responsibility and purpose.

3) *Findings*: All the codebases shown signs of Concern Overloading [11], the issues are however bigger as the codebase increased. In the case of the apps which were using Coordinating Controllers (E-Commerce, Trust, Firefox) the issues were lower than in the case of Wikipedia where the View Controller classes were way more complex as they also had to handle navigation logic.

IV. CONCLUSIONS

By our study we have tried to provide interesting insights about several common problems of MVC for both mobile developers and scientific community which are commonly found in open-source or private projects. We describe these problems in detail as well as their corresponding architectural smells. Furthermore, several solutions to those problems have been proposed which shed some light on architectural corner cases which were less explored by practitioners.

As we have shown previously in this paper, MVC can be used as the presentational software architecture for a mobile application. If the concepts are implemented correctly this does not produce any of the popular issues, neither massive view controllers nor the violation of the single responsibility principle.

What is important to be understood is that based on the complexity of the application the entities in the MVC architecture should be more granular, in order to be flexible, testable and maintainable. Based on this complexity, new types of layers or sublayers can appear which are close related to the requirements of the application.

Based on the observations made throughout many years of developing commercially those kinds of applications, the presentational architectural concept used was never an issue for the flexibility, extensibility and testability of the application; the issue always came from its bad implementation, or the misuse of programming language features.

Our further work will continue on developing tools for ensuring that a certain architectural pattern or certain architectural rules are respected with every commit made by a developer. By following this direction we can educate developers regarding the architectural aspects of a mobile software application, we will help them produce cheaper and cleaner code.

REFERENCES

- [1] GSMA. (2017) Global mobile trends. link.
- [2] A. Cowkur. (2017) Functional programming for Android developers. link.
- [3] ObjC.io. (2016) Functional programming. link.
- [4] E. Bessarabova. (2017) MVP vs MVC vs MVVM vs VIPER. What is better for iOS development? link.
- [5] K. Kocsis. (2018) Architectural patterns, MVC, MVVM: What is the hype all about? link.
- [6] E. Maxwell. (2017) MVC vs. MVP vs. MVVM on Android. link.
- [7] H. Bagheri, J. Garcia, A. Sadeghi, S. Malek, and N. Medvidovic, "Software architectural principles in contemporary mobile software: from conception to practice," *Journal of Systems and Software*, vol. 119, pp. 31–44, 2016.
- [8] D. M. Le, C. Carrillo, R. Capilla, and N. Medvidovic, "Relating architectural decay and sustainability of software systems," in *Software Architecture (WICSA), 2016 13th Working IEEE/IFIP Conference on*. IEEE, 2016, pp. 178–181.
- [9] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Identifying architectural bad smells," in *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*. IEEE, 2009, pp. 255–258.
- [10] —, "Toward a catalogue of architectural bad smells," in *International Conference on the Quality of Software Architectures*. Springer, 2009, pp. 146–162.
- [11] D. M. Le, D. Link, A. Shahbazian, and N. Medvidovic, "An empirical study of architectural decay in open-source software," in *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2018, pp. 176–17609.
- [12] R. Mo, J. Garcia, Y. Cai, and N. Medvidovic, "Mapping architectural decay instances to dependency models," in *Proceedings of the 4th International Workshop on Managing Technical Debt*. IEEE Press, 2013, pp. 39–46.
- [13] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software engineering notes*, vol. 17, no. 4, pp. 40–52, 1992.
- [14] P. Velasco-Elizondo, L. Castañeda-Calvillo, A. García-Fernández, and S. Vázquez-Reyes, "Towards detecting MVC architectural smells," in *International Conference on Software Process Improvement*. Springer, 2017, pp. 251–260.
- [15] M. Aniche, G. Bavota, C. Treude, A. Van Deursen, and M. A. Gerosa, "A validated set of smells in Model-View-Controller architectures," in *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 2016, pp. 233–243.
- [16] K. Sokolova, M. Lemercier, and L. Garcia, "Towards high quality mobile applications: Android passive MVC architecture," *International Journal On Advances in Software*, vol. 7, no. 2, pp. 123–138, 2014.
- [17] Apple. (2012) Model-View-Controller. link.
- [18] A. Sinhal. (2017) MVC, MVP and MVVM design pattern. link.
- [19] M. Potel, "MVP: Model-View-Presenter the taligent programming model for C++ and Java," *Taligent Inc*, p. 20, 1996.
- [20] R. Eckstein. (2013) Java SE application design with MVC. link.
- [21] D. Plakalovic and D. Simic, "Applying MVC and PAC patterns in mobile applications," *arXiv preprint arXiv:1001.3489*, 2010.
- [22] M. J. Yuan, *Enterprise J2ME: developing mobile Java applications*. Prentice Hall Professional, 2004.
- [23] G. E. Krasner, S. T. Pope *et al.*, "A description of the Model-View-Controller user interface paradigm in the Smalltalk-80 system," *Journal of object oriented programming*, vol. 1, no. 3, pp. 26–49, 1988.
- [24] S. M. Alam. (2017) VIPER design pattern for iOS application development. link.
- [25] Wikimedia. (2018) Wikipedia iOS application. link.
- [26] Mozilla. (2018) Firefox iOS application. link.
- [27] Trust. (2018) Trust wallet iOS application. link.
- [28] R. Garofalo, *Building enterprise applications with Windows Presentation Foundation and the Model View View Model Pattern*. Microsoft Press, 2011.
- [29] ObjC.io. (2014) Architecting iOS apps with VIPER. link.
- [30] Microsoft. (2013) ASP.NET MVC overview. link.
- [31] M. Fowler. (2006) GUI architectures. link.
- [32] R. C. Martin, "Design principles and design patterns," *Object Mentor*, vol. 1, no. 34, p. 597, 2000.
- [33] Apple. (2012) Controller. link.

An Empirical Study on Managing Energy and Accuracy Requirements of Location Based Android Applications

Marimuthu C¹

Sanjana Palisetti²

K. Chandrasekaran³

Department of Computer Science and Engineering,

National Institute of Technology Karnataka, Mangalore - 575 025, INDIA

E-mail: cs15fv08.muthu@nitk.edu.in¹, sanjana.palisetti@gmail.com², kchnitk@ieee.org³

Abstract

The improper use of GPS and location-related APIs may result in abnormal battery drain in Android applications. Over the last few years, the developers' discussions on improving energy efficiency have been increased. In this paper, we mine StackOverflow to analyze and summarize the characteristics of developers' discussions of managing energy and accuracy-related requirements of location-based Android applications. We extracted 11,911 questions from StackOverflow and filtered 320 relevant questions to answer four research questions. We conducted a manual thematic analysis on relevant questions. Our study shows that the developers are concerned about energy consumption, but are unclear about their preferences as energy and accuracy evolved as conflicting requirements.

1 Introduction

In Location Based Applications (LBAs), the improper use of location-related APIs may result in abnormal battery drain [1]. There has been an increasing amount of research efforts [2, 3] made by researchers to reduce the energy consumption of location sensing in Android smartphones. These studies have less information on the programming knowledge required to manage the energy and accuracy-related issues of LBAs. In recent years, mining StackOverflow [4, 5] to summarize the knowledge on energy consumption of Android applications proved to be a successful technique. However, these studies are not specifically focusing on location-based Android applications. Hence, in this paper, we aim to summarize the characteristics of developers' discussion of managing energy and accuracy-related requirements of location-based Android applications. More specifically, this paper aims to explore and answer the following questions:

- **RQ1:** Do developers ask questions to improve energy and accuracy requirements?
- **RQ2:** Which developer goal yield more successful answers?
- **RQ3:** Which developer goal is popular among the developer community?
- **RQ4:** How have the developers' goals changed over time?

We followed the methodology suggested by Braun et al. [6] to conduct a thematic analysis of the selected questions. We extracted 11,911 questions from StackOverflow and through semi-automated filtering and manual validation process, we identified 320 relevant questions for further analysis. We identify 3 main categorizations, namely: Status of question (*successful, unsuccessful, ordinary*), Goals of developers (*energy saving, improving accuracy, balancing both*) and Type of question (*implicit, explicit, fundamental*). The important finding of this study is that the developers are facing problem in balancing energy and accuracy-related requirements. In addition to this, the study found that several developers are unclear about their preference for selecting energy saving or improving accuracy as their primary goal.

The rest of the paper is structured as follows: Section 2 presents related works, Section 3 describes the empirical study methodology, Section 4 presents the answers to the research questions, Section 5 briefs the validity threats and Section 6 concludes the paper with possible future works.

2 Related Works

In recent years, there has been an increasing amount of empirical studies on the energy consumption of Android applications [7, 8, 4, 5]. The first empirical study on categorizing the energy-related issues of smartphones was published by Pathak et al. [8] by mining four online forums and presented a comprehensive study of energy-related issues in smartphones. Pinto et al. [4] mined StackOverflow for software energy consumption-related questions and answers and identified seven causes for energy consumption problems. Malik et al. [5] explored the quantitative and qualitative aspects of energy-related questions specific to the Android platform on StackOverflow. The authors have summarized energy-related issues into four main categories and explore the APIs that are significantly discussed in the energy-related posts. Though there have been several studies on energy consumption in Android applications, none of the studies have explicitly focused on location-based Android applications. In this paper, we aim at summarizing the developers' discussion on energy and accuracy-related issues of location-based Android applications.

3 Study Methodology

We collected the energy and accuracy-related questions of location-based Android applications using suitable SQL Queries on StackExchange Data Explorer¹. We used the keywords *android*, *location*, *gps* on the *tag* field of StackOverflow questions. The SQL query returned a total of 11,911 questions in the form of *.csv* file. This file was our raw data set which was used for the further filtering process. The dataset contains information about *Title*, *Body*, *Accepted Answer*, *Score*, *Views Count*, *Favorites Count*, *Created Date* and other relevant information. The second step is a semi-automated method to filter the questions that are specific to energy and accuracy-related issues of location-based Android applications. During this filtering process, we used keyword matching on the *Body* field of the questions and obtained 651 relevant questions². We further categorized the questions under three different categories: *successful*, *ordinary* and *unsuccessful*. The questions with negative and zero scores were removed from successful and ordinary categories as they were insignificant to our study. This reduced the dataset to 399 relevant questions. We manually read the title and body fields of the questions to verify its relevance to energy and accuracy-related requirements during which 79 false positives were found and removed resulting in 320 questions being considered for thematic analysis. We followed the guidelines given by Braun et al. [6], to conduct the thematic analysis. We defined three major themes based on the codes that we created during our reading. The first category of theme is about the type of questions asked (*Implicit*, *Explicit* and *Fundamental*), the second category of theme is based on the developers' goal (*Energy Saving*, *Improving Accuracy* and *Balancing Both*), and the third category of theme is based on the status of the question (*Successful*, *Ordinary*, *Unsuccessful*). Each question's title and body field was read and marked by the first author and verified by the second and third author. During the discussion sessions, we resolved the conflicts in the coding process. Due to space restriction, more details on the data analysis have been discussed in Section 4.

4 Answers and Discussions

In this section, we describe the analysis method and answers to the mentioned research questions.

4.1 RQ1: Do developers ask questions to improve energy and accuracy requirements?

The purpose of this research question is to know how clear the developers are with energy and accuracy-related requirements. We conducted a manual thematic analysis on the title and body field of the relevant questions. We identified the following themes:

- **Explicit:** We categorized the questions under this category, if the developers can clearly specify their requirements in terms of energy, or accuracy, or balancing both. Ex: "*Battery dies quickly when GPS or Wi-Fi is used. How can I save battery life? Is it right to request location updates every 5 seconds in the code?*" (PostID: 28407944).
- **Implicit:** We categorized questions under this category if the developers are unsure about the solutions and mentioned only the issue they are facing rather than their specific requirements or goals. Ex: "*How to keep an application running in the background? Keep collecting data?*" (PostID: 6291729)
- **Fundamental:** We categorize the questions under this category if the developer wants to gain knowledge about the topic. Ex: "*Does anyone know whether the Android addProximityAlert on the LocationManager is battery intensive?*" (PostID: 1113606).

As a result of quantitative analysis, we found 35% questions under *Explicit* category, 16% questions under *Implicit* category, and 49% questions under *fundamental* category. From the results, we can infer that the developers do ask questions to improve energy and accuracy requirements. To be more specific, 51% of the developers that ask questions about location-based Android applications strive to improve energy and accuracy requirements. Among this 51% of questions, 35% are explicit about their requirements, whereas, the rest 16% are implicit or indirect. We also infer that 49% of the developers asking fundamental questions are unaware of the best practices related to energy consumption and increasing accuracy.

4.2 RQ2: Which developer goal yield more successful answers?

The purpose of this question is to summarize which developers goals get more preference and consequently get more accepted answers. The fundamental questions (157) which were identified from the previous thematic analysis were removed as they were not contributing to either improving energy-efficiency or accuracy. The remaining 163 questions related to developers goals category were considered for thematic analysis. We identified the following themes: *Energy saving*, *Improving accuracy* and *Balancing both*. Table 1, describes the identified themes and few examples from StackOverflow. Further, we classified these 163 questions based on their status. Under the status category, we found the following themes: *successful*, *ordinary* and *unsuccessful*. The questions are classified under *successful* category if there is an *accepted* answer. The ordinary questions are categorized so if they have been answered, but none of the answers were *accepted* by the developer who posted the question. The questions with no

¹<https://data.stackexchange.com/>

²<https://bit.ly/2E1kbID>

Table 1: Identified themes under developers' goals category and examples

Goal	Description	Percentage	Example
Energy Saving	Questions were categorized under this category, if they are about reducing the battery usage by ignoring accuracy	58.89%	"How to receive driving start and stop activity with Android in an energy efficient manner which works even offline?" (PostID:45739938).
Improving Accuracy	Questions were categorized under this category, if they are about increasing the accuracy by ignoring battery draining behaviour	28.83%	"How to get location updates while device is powered?" (PostID:45204188).
Balancing Both	Questions were categorized under this category, if they are about balancing energy and accuracy related requirements	12.26%	"Android best way to get location repeatedly in background considering battery as well?" (PostID:27313684).

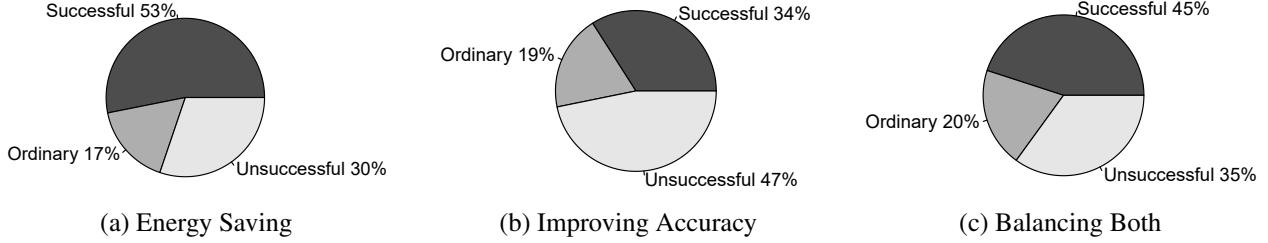


Figure 1: Developer goal versus Status of questions

answers are categorized as *unsuccessful*. The categorized questions were grouped based on the status of the question and the developer's goal.

The result of thematic analysis is represented in Figure 1 where we observe that *energy saving* related questions yield the most successful answers, closely followed by questions related to *balancing both*. This shows that the developers are more interested in either energy saving or balancing both conflicting requirements (energy and accuracy). On the other hand, *improving accuracy* alone has not been given much preference as it yields 47% *unsuccessful* questions. This shows that improving accuracy by ignoring energy issues may not be feasible or not a best practice while developing location based Android applications. Therefore, to build a successful application we believe that balancing between both the conflicting requirements is the essential factor.

4.3 RQ3: Which developer goal is more popular among the developer community?

We followed the method suggested by Pinto et al. [4] to calculate the popularity of the questions. The formula for calculating the popularity of the questions is as follows:

$$P = S + A + C + F + V \quad (1)$$

Where, P is the calculated popularity of the question, S is the score of the question, A is the number of answers, C is the number of comments, F is the number of favoritizations, and V is the number of views. Here, we slightly deviated from the guidelines suggested by Pinto et al. [4] with respect to selecting StackOverflow views. Here, instead of selecting overall StackOverflow views, we used the average views of all questions related to location based android applications. Hence, the views of a question can be calculated

Table 2: Popularity of questions under each category

Developer Goal	Median (V)	Median (P)
Energy Saving	643.5	6.039
Improving Accuracy	373	5.047
Balancing Both	523	6.755

as follows:

$$\frac{\text{Views of question } (Q)}{\text{Avg. views of all LBAs related questions}} \quad (2)$$

As shown in Table 2, it is clear that the theme of *balancing both* energy and accuracy has gained more popularity than the rest. Although the number of questions related to balancing both requirements is the least, this category has scored the most popularity, reflecting the importance of giving equal priority to both types of requirements. Further, *energy saving* is observed as the next most popular requirement. Activities such as continuous background location updates and improper use of location APIs result in more abnormal battery drain, making it a higher concern of the developers to extend users battery life.

4.4 RQ4: How have the developers' goals changed over time?

The purpose of this question is to observe the trend in developers preference towards *energy saving*, or *improving accuracy*, or *balancing both*. The obtained information related to questions were grouped by the year and the occurrences of each goals category were tabulated. The data were quantitatively analyzed and the trend over the years is depicted in Figure 2. As shown in Figure 2, we observe that energy saving has been given a higher priority from the beginning in 2010 with a ratio of 3:0:0 questions in each category (EnergySaving:ImprovingAccuracy:BalancingBoth). But as the years passed the priority of saving energy rela-

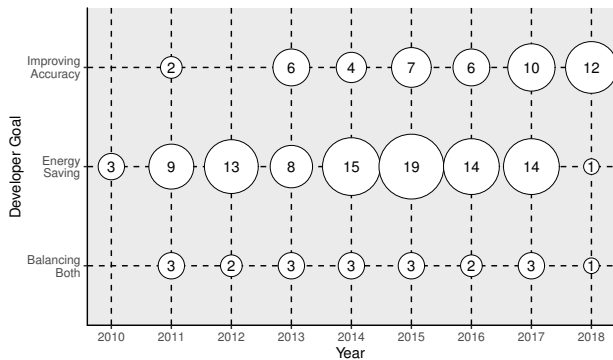


Figure 2: Developers goals over time.

tively decreased, changing the ratio to 1:12:1 in 2018. Till 2014, Android-native APIs were used, making it the task of the developers to configure everything by themselves resulting in major energy consumption issues. However, in 2015, the introduction of Google Location APIs had decreased the energy consumption related issues, hence, decreasing the requirement of the developers to find solutions for abnormal energy consumption. In the perspective of accuracy-related questions, developers have never shown much concern until 2016. However, in the years of 2017 and 2018, the interest in accuracy-related questions drastically rose due to the introduction of Doze mode in higher versions of Android. Doze mode, in short, kills the background location updates when a smartphone enters sleep mode. Users on StackOverflow have complained about not being able to decrease the time between consecutive background service updates and receiving the following message, “In an effort to reduce power consumption, Android 8.0 (API level 26) limits how frequently background apps can retrieve the user’s current location. Apps can receive location updates only a few times each hour” (PostID: 47471600). This resulted in lesser accuracy as continuous location updates were not possible. Hence, developers were more interested in solving accuracy-related issues in recent years. As shown in Figure 2, we also observe that the questions related to *energy saving* decrease drastically to 1 in 2018 from a maximum of 19 in 2015 while there was an increase in accuracy-related questions. Hence, energy and accuracy evolve as two conflicting requirements, polarizing the interests of the developers.

5 Threats to Validity

This section presents the internal and external threats to validity.

Internal: As the scope of the study is narrowed down to location-based Android applications, we selected only 320 questions for thematic analysis. However, we believe that these manually selected questions may be a better candidate set for summarizing the characteristics of developers discussions. Second, the presence of false positives in the

dataset. To reduce the number of false-positives, we conducted manual in-depth reading on all filtered questions and carefully removed the false positives.

External: First, our results are only limited to Questions and Answers on StackOverflow. Other online forums, online surveys, and physical interviews have not been used to obtain information. Second, the solutions and results presented cannot be generalized to other software, domains or type of developers. They are only applicable to location-based Android applications.

6 Conclusion and Future Works

In this paper, we present the results of an empirical study summarizing the characteristics of developers’ discussions on StackOverflow about energy and accuracy requirements in location-based Android applications. We identify three main categorizations, namely: Status of the question, Goals of developers and Type of question. We applied quantitative and thematic analysis to answer four research questions based off 320 relevant StackOverflow questions. As a result of this study, we were able to infer that developers on StackOverflow are facing both energy and accuracy-related issues, however, are finding it difficult to balance both as energy and accuracy requirements have evolved as conflicting requirements. As future work, we aim to analyze the answers of relevant questions to provide more qualitative insights on balancing the energy and accuracy requirements of location-based Android applications.

References

- [1] N. Capurso, T. Song, W. Cheng, J. Yu, and X. Cheng, “An android-based mechanism for energy efficient localization depending on indoor/outdoor context,” *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 299–307, 2017.
- [2] D. Kim, S. Lee, and H. Bahn, “An adaptive location detection scheme for energy-efficiency of smartphones,” *Pervasive and Mobile Computing*, vol. 31, pp. 67–78, 2016.
- [3] T. Choi, Y. Chon, and H. Cha, “Energy-efficient wifi scanning for localization,” *Pervasive and Mobile Computing*, vol. 37, pp. 124–138, 2017.
- [4] G. Pinto, F. Castor, and Y. D. Liu, “Mining questions about software energy consumption,” in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 22–31.
- [5] H. Malik, P. Zhao, and M. Godfrey, “Going green: An exploratory analysis of energy-related questions,” in *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 2015, pp. 418–421.
- [6] V. Braun and V. Clarke, *Successful qualitative research: A practical guide for beginners*. sage, 2013.
- [7] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, “What do programmers know about software energy consumption?” *IEEE Software*, vol. 33, no. 3, pp. 83–89, 2016.
- [8] A. Pathak, Y. C. Hu, and M. Zhang, “Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices,” in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011, p. 5.

LAD: A Layout Anomaly Detector for Android Applications

Cheng-Zen Yang, Chih-Ju Lai, Peng Lu, and Zhi-Jun You

Department of Computer Science and Engineering

Yuan Ze University

Chungli, Taiwan, R. O. C.

{czyang,cjl16,pl16,zjy19}@syslab.cse.yzu.edu.tw

Abstract— In recent years, Android has become one of the most popular mobile operating systems. Software testing of Android applications becomes a very important issue. In the past research, many studies focus on functional testing. Recently, some studies start to discuss the layout anomaly issue. However, these schemes investigate few types of layout anomalies. Moreover, there may exist underlying platform constraints. In this paper, a detection tool LAD (Layout Anomaly Detector) is proposed. LAD considers the anomaly types studied in the previous work with a newly proposed anomaly type for scale maladaptation of components and text. LAD also supports testing scripts. We have conducted empirical experiments with four real apps. The experimental results show that LAD can effectively detect the layout anomalies of these apps.

Keywords—Android; layout anomaly; GUI layout testing; automatic testing

I. INTRODUCTION

As shown in the statistics of AppBrain, the number of Android apps has exceeded 2.59 million on Google Play in March 2019 [1]. In the past, automatic Android software testing has received many attentions, such as [2-5]. However, most of the previous studies focus on functional testing. To our best knowledge, the number of research studies on detecting layout anomalies for Android apps is comparatively few [6-9]. Because Android devices have the severe fragmentation problem, layout anomalies become an important issue for user experience.

The layout anomaly problem occurs in an app if its layout design does not consider the diversity of different screen specifications. This problem can be very annoying for users and app developers. For example, the layout anomalies such as UI component overlap or cropped text may hamper the user operations. As reported in [8], some UI components may even disappear when the app is executed in a smaller screen size.

To tackle layout anomalies, app developers need to take a lot of time to inspect the layouts for many screen specifications. As illustrated in [8, 9], if there are m layouts in an app and n screen specifications, the time complexity of the inspection work is $O(m \times n)$. This workload is harmless if n is very small. However, a survey of OpenSignal shows that over 150 different screen sizes have existed for Android devices in 2014 [10]. If there are 100 different screen specifications to be inspected and the inspection time is 5 minutes per layout, the total inspection time will be 83 hours for an app of 10 layouts.

In recent years, studies have been conducted to deal with the problem of layout anomaly detection. Several tools like Galen Framework [6] and ITArray Automotion Framework [7] are developed to perform visual testing for Android apps. Their users need either to write scripts to specify the geometry relationships or to import the customized Java code into the app under testing (AUT). In 2017, Shih *et al.* proposed an automatic detection tool called UI-Explorer to detect two types of layout anomalies [8]. In UI-explorer, all layouts are traversed and inspected automatically. However, UI-explorer provides only click events to traverse the layouts. It lacks an approach to handle user inputs. Hasellknippe and Li also proposed a tool called the Layout Bug Hunter (LBH) to detect three different types of layout anomalies [9]. LBH does not need any user specification. However, LBH relies on the underlying Fuse platform [11] to implement the GUI analytical work.

In this paper, we propose a layout anomaly detection tool called LAD (Layout Anomaly Detector) to perform visual testing on Android app layouts. LAD is designed to detect six types of layout anomalies: *components missing*, *cropped text*, *component overlap*, *component overflow*, *component misalignment*, and *component/text scale maladaptation*. To automate the detection process, a procedural script language is devised to specify the navigation procedure and the detection operations. LAD also supports two inspection modes: the *standalone* mode and the *comparative* mode. In the comparative mode, users specify a designated resolution to get *gold standard layouts* (GSLs) as the inspection templates. LAD automatically inspects all *layouts under test* (LUTs) in the testing resolutions against the GSLs. In the standalone mode, LAD works only for one testing resolution. To operate in a contiguous integration (CI) testing process, LAD is designed as a native Java tool in a modular design.

The rest of this paper is organized as follows. Section 2 briefly reviews the related research work on the layout anomaly detection problem. Section 3 describes the layout anomalies detected in LAD and the LAD design. Section 4 presents the empirical study on LAD with four Android apps. Finally, Section 5 concludes this paper.

II. RELATED WORK

The layout anomaly detection problem is an important issue for traditional window-based interface design and Web application design [12-14]. For automatically detecting layout

anomalies, the detection approaches can be mainly classified into two categories: the image comparing approach [15, 16] and the layout analysis approach [6-9].

The image comparing approach inspects the layout anomalies by making comparisons between two captured screenshots to find their pixel differences. Since there is no layout structure information in the contrasting process, this approach may suffer from the challenge to decide whether two images presented in difference screen sizes/resolutions are identical [9].

For the layout analysis approach, the layout information is used to detect the anomalies. In Galen Framework [6], users can program the test cases using Galen Specs Language to describe the relationships between two objects. However, Galen is mainly designed for Web apps. In ITArray Automation Framework [7], a library is supported to perform visual testing on web and mobile pages. Users can describe the visual relationships using the library calls. As pointed in [9], however, these tools require users to provide detailed information about the placement of GUI objects and the validation parameters. If the GUI layouts are changed frequently, this scripting-in-detail approach may incur high maintenance costs.

In 2017, Shih *et al.* proposed a tool called UI-Explorer to automate the detection of layout anomalies [8]. UI-Explorer uses a GUI ripping engine to extract the layout information of all GUI components and automatically traverse all layouts under two different screen specifications. With UI-Explorer, users do not need to provide detailed visual information for testing. However, UI-Explorer considers only two types of layout anomalies: *components missing* and *cropped text*. Moreover, it provides only click events to traverse the layouts.

Hasellknippe and Li proposed a tool called the Layout Bug Hunter (LBH) for detecting three GUI layout anomalies: *component overlap*, *component overflow*, and *component misalignment* [9]. LBH employs a set of specified rules to detect layout anomalies. Therefore, LBH does not require any script for its visual testing. However, LBH does not consider the comparative approach. As discussed in [9], an intentional overlap design may be identified as an anomaly. Moreover, the current LBH design relies on the underlying Fuse platform [11].

III. DESIGN OF LAYOUT ANOMALY DETECTION MECHANISMS

This section first presents the layout anomalies considered in LAD. The design of the LAD architecture is described thereafter.

A. Layout Anomalies

In addition to the layout anomaly types discussed in the previous studies [8, 9], *components missing*, *cropped text*, *component overlap*, *component overflow*, and *component misalignment*, LAD considers one more type: *component/text scale maladaptation*. LAD supports two inspection modes for layout anomaly detection: the *standalone* mode and the *comparative* mode. In the *standalone* mode, LAD uses the pre-defined heuristic rules to decide the detection results. In the *comparative* mode, the layouts in a user-designated resolution are used as the inspection templates, which are the *gold standard*

layouts (GSLs). The *layouts under test* (LUTs) in different screen sizes/resolutions will be investigated against the GSLs.

1) Component Missing

A component missing anomaly happens when some GUI component disappears in a different LUT screen size/resolution. Generally, this anomaly occurs when the screen size/resolution becomes smaller. As shown in Figure 1, the gender symbol at the bottom of the app in the 768×1280 GSL disappears in the 480×800 LUT.

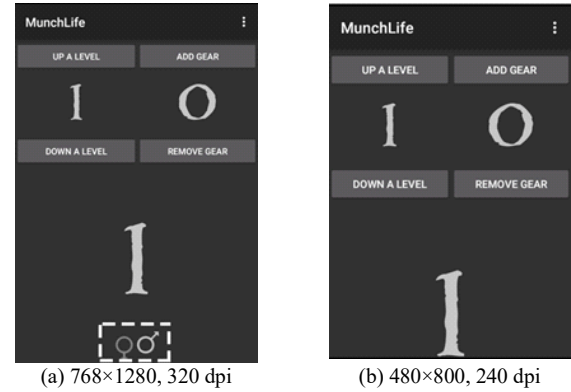


Figure 1: Component missing anomaly

To detect component missing anomalies, LAD first retrieves the XML dumps of GSLs and LUTs. From the XML dumps, LAD calculates the total numbers of components of each GSL and the corresponding LUT. If these two numbers are not equal, the LUT is regarded as having a component missing anomaly.

2) Cropped Text

The cropped text anomaly happens when some text string correctly appears in the GSL, but it is cropped in the corresponding LUT. Generally, this anomaly occurs because the GUI components are squeezed in the LUT of a smaller screen resolution. As shown in Figure 2, when the 768×1280 GSL resolution is shrunk to the 480×800 LUT resolution, the text in the red box of Figure 2(b) is cropped. In this case, the app user cannot read the complete text.

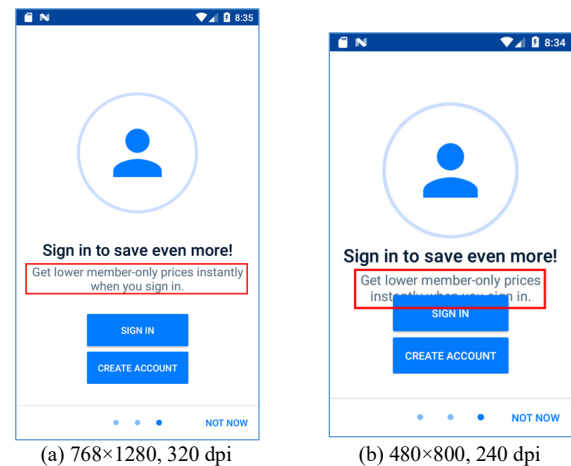


Figure 2: Cropped text anomaly.

LAD uses the same approach proposed in [8] to first obtain the coordinates of the text component C_1 in the GSL and invoke OpenCV (a computer vision library) to get a screenshot S_1 of C_1 . LAD then invokes Tesseract (an OCR text recognizer) to perform the OCR work and get the OCR text T_1 . Similarly, LAD gets the OCR text T_2 for the corresponding component C_2 in the LUT. For example, T_1 in the red box of Figure 2(a) is recognized as “Getlowermember-onlypricesinstantlywhenyousignin” and T_2 is recognized as “GetlowermemberonlypricesWMin-”.

Because the OCR results usually have errors, LAD calculates the Levenshtein distance (Ed_1) between T_1 and the original string $C_1.text$. LAD also calculates Ed_2 between T_2 and $C_1.text$. Therefore, if $|Ed_1 - Ed_2| \geq T_{ED}$ where T_{ED} is a predefined threshold, the LUT is regarded as having a cropped text anomaly.

3) Component Overlap

The component overlap anomaly happens when one GUI component covers a part of another GUI component. This may occur when a smaller screen size/resolution is used for LUTs. However, the original design may also contain this anomaly. Therefore, LAD detects it in both inspection modes.

Figure 3 shows an example in which Figure 3(a) is a correctly displayed GSL of a calculator app. In Figure 3(b), the dashed button area overlaps the background text area in the LUT. In this case, the app users cannot get the correct result.

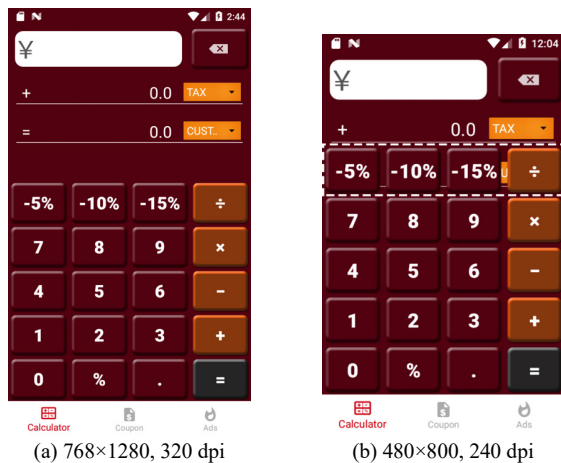


Figure 3: Component overlap anomaly.

To detect component overlap anomalies, LAD retrieves the XML dumps of LUTs and obtains the coordinates of the upper left corner and the lower right corner of each component C_i from the XML dumps. Then LAD calculates the areas of two components to see if these two areas are overlapped.

4) Component Overflow

When an app developer forgets to adjust the component size according to the new LUT resolution, a component overflow anomaly may occur in the LUT if the area of some component exceeds the screen boundary. In the worst case, users may not be able to correctly operate the component.

Figure 4 shows an example. In the 768x1280 GSL of Figure 4(a), both button components are completely displayed. In

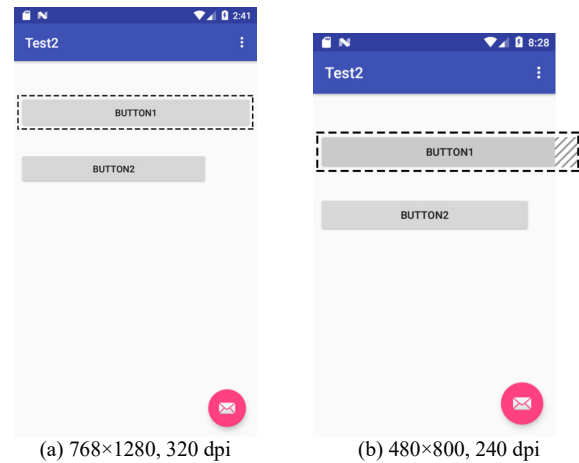


Figure 4: Component overflow anomaly.

Figure 4(b), however, the area of BUTTON1 exceeds the boundary of the 480x800 LUT layout.

LAD also analyzes the XML dumps of LUTs to obtain the coordinate (x_{i1}, y_{i1}) of the upper left corner of each component C_i and its coordinate (x_{i2}, y_{i2}) of the lower right corner. LAD also analyzes the component information to get the height H_i and the width W_i . LAD compares these values with the screen height S_h and the screen width S_w , respectively. If one of the following inequalities is satisfied, the LUT is regarded as having a component overflow anomaly.

- Horizontal boundary overflow

$$x_{i1} + W_i > S_w \parallel x_{i2} < W_i \parallel x_{i2} > S_w$$

- Vertical boundary overflow

$$y_{i1} + H_i > S_h \parallel y_{i2} < H_i \parallel y_{i2} > S_h$$

5) Misalignment

When a programmer designs a layout, the components are usually aligned in the layout. There are three kinds of vertical alignment: *Align Top*, *Align Middle*, and *Align Bottom*. Similarly, the horizontal alignment also has three kinds: *Align Left*, *Align Center*, and *Align Right*. When the component position is not properly adjusted with the resolution change, it is possible that a

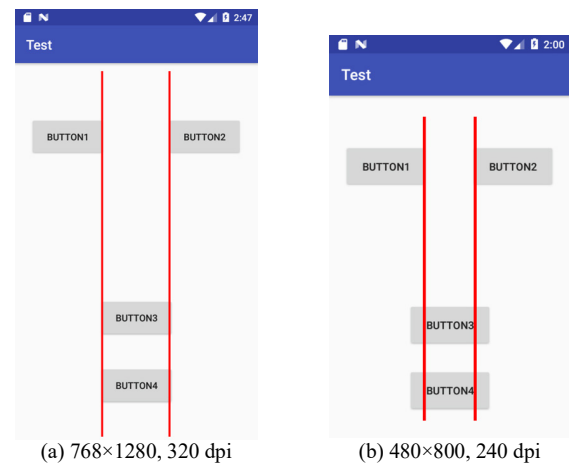


Figure 5: Component misalignment anomaly.

component misalignment anomaly occurs in the LUT resolution. For example, the 768×1280 GSL in Figure 5(a) illustrates that the left sides of BUTTON3 and BUTTON4 are aligned with the right side of BUTTON1 along with the red line. However, the 480×800 LUT in Figure 5(b) demonstrates that BUTTON1 is not horizontally aligned with BUTTON3 and BUTTON4.

To detect component misalignment anomalies, LAD makes comparisons for each pair of two components C_i and C_j using their coordinates and centers. However, a strict alignment judgement may incur many false alarms for two cases:

- Some very minor misalignments can be allowable because it is very hard for users to find them by eyes.
- Some misalignments are intentionally designed.

Therefore, LAD has two thresholds D_S and D_L to control the misalignment detection. D_S is used to ignore the misalignments that are not easily recognized by users. In the current LAD design, the default value of D_S is 2 pixels. D_L is used to ignore the misalignments that are intentionally designed. In LAD, the default value of D_L is 8 pixels. For example, if the left upper corner of C_i is (x_{i1}, y_{i1}) and the left upper corner of C_j is (x_{j1}, y_{j1}) , the LUT is regarded as having a component misalignment anomaly when the following condition is satisfied.

$$D_S < |x_{i1} - x_{j1}| < D_L$$

6) Component/Text Scale Maladaptation

As the resolution of the screen changes, the sizes of the components and text characters may also change. If the size of a component is excessively reduced, users will have difficulty clicking the component or reading the text. The LUT is regarded as having a component/text scale maladaptation anomaly. LAD tackles this scale maladaptation problem separately for GUI components and text strings.

For GUI components, the size of an interactive component should be at least 44 pt × 44 pt according to the Apple User Interface Design Guide [17]. In [18], however, Zea suggests that the minimum target size of the component should be 5 mm × 5 mm = 30 px × 30 px. In the current LAD design, we use the size suggested by Zea as the threshold to determine the scale maladaptation anomaly for a component. If the size of a component is smaller than this threshold, a scale maladaptation

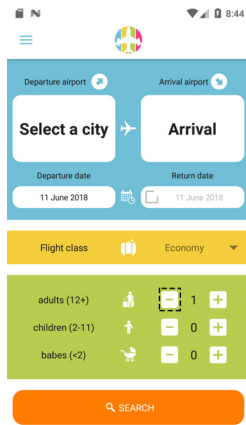


Figure 6: Component scale maladaptation anomaly in 768×1280 LUT.

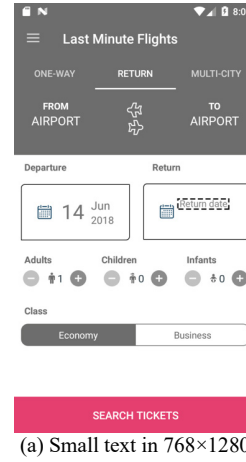


Figure 7: Text scale maladaptation anomaly.

anomaly occurs on the component. For example, Figure 6 illustrates that the dashed area is smaller than 5 mm × 5 mm in the 768×1280 LUT.

For the text scale problem, Kahn and Lenk have pointed out that the font sizes of 12 and 14 points allow the user to read comfortably [19]. Tennant also points out that the 16-pixel font size is approximately equal to the 12-point font size which is suitable for reading [20]. Because the height of the 12-point font is approximately 0.42 cm, LAD use this as a default threshold to determine if the scale of a text string is maladapted in the LUT.

Figure 7 illustrates an example in which LAD first gets the coordinates of the dashed box containing a string “Return date” and then invokes OpenCV to get its screenshot. LAD then uses a text edge detection algorithm [21] to obtain the height H_T of the string as shown in Figure 7(b). If $H_T < 0.42$ cm, the LUT is regarded as having a text scale maladaptation anomaly.

B. LAD Architecture

1) Script Language

To automate the visual testing, a script language is devised in LAD for controlling the testing procedure. With the test scripts, LAD can automatically enter text inputs into the

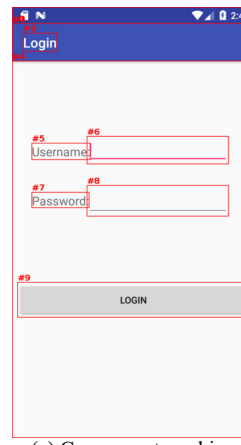


Figure 8: Component marks and the list of their IDs.

```

1: #Username
2: LAD.input[@id='editText','User']
3: #Password
4: LAD.input[@id='editText2','1234']
5: LAD.testoverlap
6: button.click
7: LAD.testall

```

Figure 9: An example script.

components and navigate to different layouts. In order to let the script writer know the component IDs of each layout, LAD has a companion tool LADMark to extract the component IDs and mark each component on the layout screenshot as shown in Figure 8(a). Figure 8(b) is the component list.

Figure 9 shows a script example based on the list in Figure 8(b). A starting “#” is used to denote a comment. On the second line, “LAD.input[@id='editText','User']” is used to enter the text input “User” for the username. On line 4, “LAD.input[@id='editText2','1234']” is used to input the password “1234”. On line 5, LAD.testoverlap instructs LAD to perform anomaly detection for component overlap. On line 6, button.click tells LAD to click the LOGIN button such that the app moves to the next layout. Finally, LAD.testall instructs LAD to perform anomaly detection for all kinds of anomalies.

2) Architecture Design

Figure 10 shows the architecture of the LAD. The system reads the AUT, the LUT screen resolution, the GSL screen resolution, and the testing scripts. Then, LAD analyzes the testing scripts and automatically navigates the GSLs and LUTs for testing.

Finally, LAD collects the testing results and generates the detection report. To help the developer find the corresponding places of the detected anomalies, the report contains the

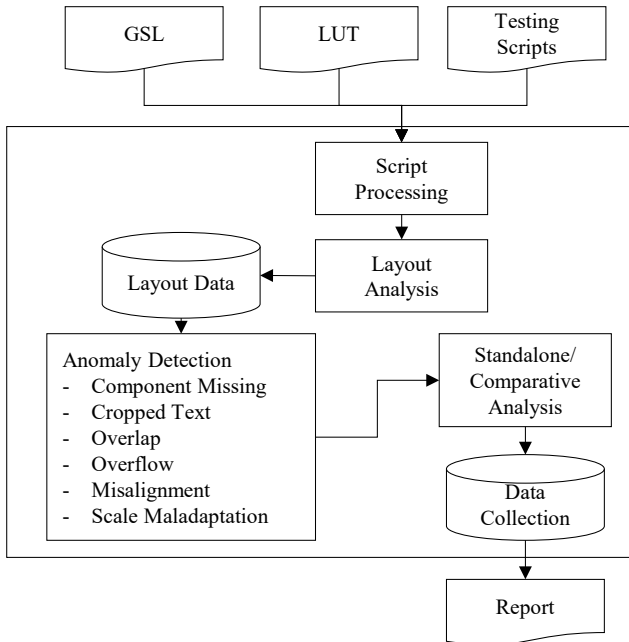


Figure 10: Detection system architecture.

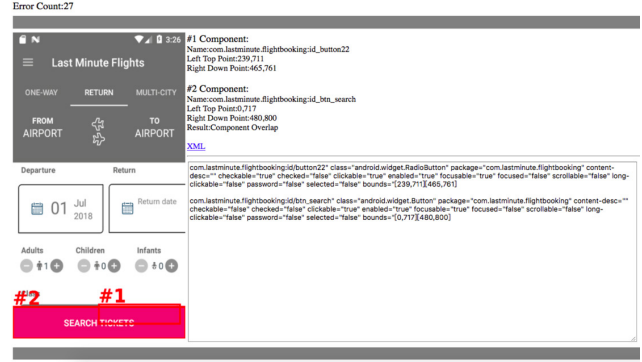


Figure 11: A detection report example.

screenshots, the detection results, and the XML information of the problematic components. Figure 11 illustrates a report example.

IV. EXPERIMENTS

To evaluate the effectiveness of the proposed anomaly detection mechanism, we have conducted experiments with four Android apps. This section describes the experimental results.

A. Experimental Environment

We have implemented a prototype for experiments. The LAD prototype was executed on a PC running with 16 GB RAM and an Intel i5-6500 3.20 GHz CPU. The operating system was Ubuntu 16.04 64bit. The Android emulator was the built-in emulator of Android Studio 3.1 emulating Google 7.0.0 API 24. The resolutions of GSL and LUT were 768×1280 and 480×800, respectively. Four investigated apps are as follows:

- **MunchLife** 1.4.4 (ML): This app is used to track the character level for a card game Munchkin.
- **Last Minute Flights Booking** 2.0 (LMFB): This app allows users to compare flight and hotel information provided by airlines and travel agencies. Users can use it to find the most suitable flights and hotels.
- **Priceline** 4.34.170 (PL): This app allows users to quickly plan their travel itineraries, including booking cheap flights, hotels and rental cars.
- **MyAir365** 1.06 (MA): This app allow users to easily order low-cost airlines.

The main reason for choosing these four apps is that they all have layout anomalies. Therefore, they can be used to demonstrate the detection performance. In order to clearly present the detection results, this paper shows the experimental results using only the first page of each app.

B. Experimental Results

As shown in Table 1, the numbers are the anomalies detected in the experiments. For anomalies of *components missing*, *cropped text*, *component overlap*, *component overflow*, and *component misalignment*, the results are obtained in the *comparative* mode. For scale maladaptation anomalies, the results are measured in the *standalone* mode with the 768×1280

TABLE I. DETECTION RESULTS OF LAD.

	ML	LMFB	PL	MA
Comp. missing	1	0	0	0
Cropped text	0	3	4	0
Overlap	2	10	0	7
Overflow	0	0	0	0
Misalignment	0	6	0	11
Comp. scale maladapt.	0	5	8	8
Text scale maladapt.	5	10	8	3

resolution. The manual inspections show that these detection results can effectively discover many anomalies which may be ignored by inspectors.

In the experiments, some false alarms were found in the standalone mode because the default heuristic thresholds are not appropriate for some apps and scree specifications. However, these false alarms can be eliminated in the comparative mode because the GSLs can filter out their detections in the LUTs.

V. CONCLUSION

Since Android has the severe fragmentation problem, visual testing has received considerable attentions in recent research. Traditionally, developers need to spend much time in manually checking layouts with different screen resolutions. This manual inspection process is time-consuming and error-prone.

In this paper, we present the design of an anomaly detection tool LAD which can be used to detect six types of layout anomalies. To automatically detect these anomalies in different layouts, a procedural script language is devised to specify the app navigation procedure and the detection operations. To operate in a contiguous integration (CI) testing environment, LAD is designed as a native Java tool in a modular design. Moreover, LAD supports two inspection modes: the *standalone* mode and the *comparative* mode. The *comparative* mode facilitates the avoidance of false alarms in the detection results.

In our future work, we plan to conduct comprehensive investigations with more apps to study the effectiveness of LAD. Human inspectors will also involve the experiments to make performance comparisons by measuring more metrics, such as accuracy, precision, and recall. Moreover, the functionality of LAD will be enhanced. The current design does not have a plug-in interface. Therefore, it is inflexible for users who want to add a new module for a new anomaly. The algorithms for anomaly detection will be also improved further and the practical applicability of the detection reports will be enhanced to help developers quickly detect the layout anomalies and improve the quality of Android apps.

ACKNOWLEDGEMENT

This work was supported in part by Ministry of Science and Technology, Taiwan under grant MOST 107-2221-E-155-012. The authors would also like to express many thanks to the anonymous reviewers for their precious suggestions.

REFERENCES

- [1] AppBrain, "Android apps on Google Play", <https://www.appbrain.com/stats/number-of-android-apps>, last accessed on March 1, 2019.
- [2] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using GUI Ripping for Automated Testing of Android Applications", in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*, pp. 258-261, 2012.
- [3] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon, "MobiGUITAR: Automated Model-Based Testing of Mobile Apps", *IEEE Software*, vol. 32, no. 5, pp. 53-59, 2015.
- [4] C. Hu, and I. Neamtii, "Automating GUI Testing for Android Applications", in *Proceedings of the 6th International Workshop on Automation of Software Test (AST '11)*, pp. 77-83, 2011.
- [5] H.-L. Wen, C.-H. Lin, T.-H. Hsieh, and C.-Z. Yang, "PATs: A Parallel GUI Testing Framework for Android Applications", in *Proceedings of the 2015 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC 2015)*, pp. 210-215, 2015.
- [6] Galen Framework, "Galen Framework", 2019, <http://galenframework.com/>, last accessed on March 1, 2019.
- [7] D. Zaiats, "ITArray Automotion Framework", 2019, <https://automotion.itarray.net/>, last accessed on March 1, 2019.
- [8] Y.-A. Shih, Y.-P. Chang, and C.-Z. Yang, "An Automated Detection Framework for Testing Visual GUI Layouts of Android Applications", in *Proceedings of the 7th International Workshop on Computer Science and Engineering (WCSE 2017)*, pp. 544-548, 2017.
- [9] K. F. Hasselknippe and J. Li, "A Novel Tool for Automatic GUI Layout Testing", in *Proceedings of the 24th Asia-Pacific Software Engineering Conference (APSEC 2017)*, pp. 695-700, 2017.
- [10] OpenSignal, "Android Fragmentation Visualized (August 2014)" <https://opensignal.com/reports/2014/android-fragmentation/>, 2014, last accessed on December 20, 2017.
- [11] Fusetools AS, "Fusetools - We Make Apps Easy", <https://www.fusetools.com>, 2019, last accessed on March 1, 2019.
- [12] E. Ch'ng, and D. C. L. Ngo, "Screen Design: A Dynamic Symmetry Grid Based Approach", *Displays*, vol. 24, pp. 125-135, 2003.
- [13] H. Song, H. Liu, and D. Chen, "An Automatic GUI Adjustment Method for Mobile Computing", in *Proceedings of the 2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE 2011)*, pp. 206-210, 2011.
- [14] S. Hallé, N. Bergeron, F. Guerin, G. L. Breton, and O. Beroual, "Declarative Layout Constraints for Testing Web Applications", *Journal of Logical and Algebraic Methods in Programming*, vol. 85, no. 5, pp. 737-758, 2016.
- [15] Applitools, "Appltools Eyes", <https://applitools.com/>, 2019, last accessed on March 1, 2019.
- [16] J. Cryer, "PhantomCSS", <https://github.com/HuddleEng/PhantomCSS>, 2019, last accessed on March 1, 2019.
- [17] Apple Inc., "Human Interface Guidelines - Adaptivity and Layout", <https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/adaptivity-and-layout/>, 2019, last accessed on March 1, 2019.
- [18] R. Zea, "Mastering Responsive Web Design with HTML5 and CSS3", Packt Publishing Ltd., 2015.
- [19] P. Kahn, and K. Lenk, "Design: Principles of Typography for User Interface Design", *Interactions*, vol. 5, no. 6, pp. 15-29, 1998.
- [20] D. B. Tennant, "16 Pixels Font Size: For Body Copy. Anything Less Is A Costly Mistake", *Smashing*, 2011, <https://www.smashingmagazine.com/2011/10/16-pixels-body-copy-anything-less-costly-mistake/>
- [21] C. Liu, C. Wang, and R. Dai, "Text Detection in Images Based on Unsupervised Classification of Edge-based Features", in *Proceedings of the 2005 8th International Conference on Document Analysis and Recognition (ICDAR '05)*, pp. 610-614, 2005.

Schedulability analysis for real-time mobile systems

Cong Chen*, Yangyang Chen*, Jian-Min Jiang*[†], Shi Zhang*, Zhong Hong*, Hongping Shu[†], and Qiong Zeng[†]

*College of Mathematics and Informatics, Fujian Normal University, Fuzhou 350007, China

[†]College of Software Engineering, ChengDu University of Information Technology, Chengdu 610103, China

Abstract—Autonomous driving systems are complex real-time mobile systems. To guarantee their safety and security, the mobile objects (agents) in these systems must be isolated from each other so that they do not collide with each other. Since isolation means two or more mobile objects cannot be located in the same area at the same time, a scheduling policy is required to control the movement of these mobile objects. However, traditional scheduling theories are based on task scheduling which is coarse-grained and cannot be directly used for fine-grained isolation controls. In this paper, we first propose an event-based formal model called a *time dependency structure* which is used to model and analyze real-time mobile systems. Then, an event-based schedule is defined. Finally, we analyze the schedulability of isolation—that is, checking whether a given schedule ensures the isolation relationship among mobile objects or not.

Index Terms—mobility, isolation, scheduling policy, ambient.

I. INTRODUCTION

Autonomous driving systems are complex mobile systems, which are a prominent subcategory of cyber-physical systems. In these complex mobile systems, safety and security, especially the isolation of mobile objects (agents), has become crucial issues. Isolation means two or more mobile objects cannot be located in the same area at the same time. The mobile objects in real-time mobile systems must be isolated from each other so that they do not collide with each other. Thus, we should have an effective mechanism for checking whether a given scheduling policy can ensure the isolation of mobile objects or not.

Existing scheduling theories, e.g., [2], [7], [8], [10]–[12], focus on task scheduling. Task scheduling mainly consider how to generate the optimal scheduling policies while the objective of the schedulability analysis is to verify that there are no violations of constraint conditions. However, as for the inherent complexity of scheduling, existing work is far from enough to solve these problems. Specifically, in the practical mobile systems, mobile objects and environments interact with each other, it is very difficult to separate a mobile system into independent tasks. We cannot directly use existing methods and techniques to obtain the scheduling policies for the isolation of mobile objects.

To solve the problem, it is necessary for a new scheduling policy to control the whole mobile system [1], [4]. Jiang et

al. [4] have proposed more fine-grained event-based scheduling instead of task scheduling. An event is generally the occurrence of an action or activity. A task, process or complex activity may consist of multiple events [6]. A complex scheduling problem cannot be decomposed into independent tasks, but it can be divided into sub-problems of event-based scheduling. Though Jiang et al. [4] have discussed event-based scheduling, such event-based scheduling does not consider real-time scheduling controls, especially the scheduling in real-time mobile systems.

To investigate the scheduling in real-time mobile systems, we must model real-time mobile systems. We extend the dependency structure model [4], [5] and add the time modeling power to it. Such a model is called a *time dependency structure*, which can conveniently specify the timing constraints and mobility of real-time mobile systems.

In this paper, we first introduce a time dependency structure. Then, an event-based schedule is defined. Finally, we investigate the schedulability analysis of isolation—that is, checking whether a given schedule ensures the isolation relationship among mobile objects or not in a real-time mobile system.

II. NOTATION AND RUNNING EXAMPLE

We will adopt the concept similar to the ambient calculus [3], where computation happens in an ambient that is a closed and bounded place and a mobile object (agent) can enter or exit an ambient.

Here, we first give some notations. Given a set X , the notations 2^X and $|X|$ denote the power set and the size of X , respectively. **Time** = $[0, \infty)$, the set of non-negative reals, denotes the domain of time. **A** and **M** denote the sets of ambients and mobile objects, respectively. The event of a mobile object $\mathcal{M} \in \mathbf{M}$ for entering an ambient $\mathcal{A} (\mathcal{A} \in \mathbf{A})$ is denoted by $en_{\mathcal{A}}^{\mathcal{M}}$ and the event of \mathcal{M} for exiting \mathcal{A} is done by $ex_{\mathcal{A}}^{\mathcal{M}}$. In fact, it is enough for us to only use the two movement events (entering and exiting events) for specifying the mobility in a mobile system. For more information, please refer to our previous work [4].

We present a running example, which is a simple yet typical mobile system where a passenger *John* needs to take a bus in a road intersection area. It is assumed that all the vehicles are equipped with Navigation Satellite System (GPS or BDS) devices and have access to a digital map database, which provide them with critical information such as position, heading, speed, road and lane details. The road area is represented as a

Corresponding author: Jian-Min Jiang (jjm@fjnu.edu.cn). This work is supported by National Natural Science Foundation of China (No. 61772004) and the NSF of Fujian province (No. 2018J01777).

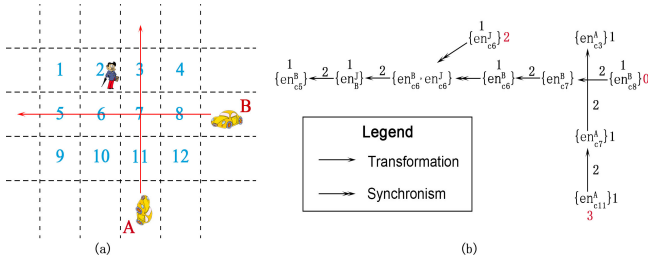


Fig. 1. A simple mobile system

grid which is divided into small cells in Figure 1.(a). Each cell in the grid is associated with a unique identifier. The buses A and B pass the intersection cell $c7$. John is located in the cell $c2$. The bus A moves along the cells $c11, c7, c3$ and the bus B does along the cells $c8, c7, c6, c5$.

John may enter the cell $c6$ and take the bus B . For simplification, we give some notations. John is denoted as J , and the event of John for entering the bus B (resp. the cell $c6$) is denoted as en_B^J (resp. en_{c6}^J). If a bus X enters a cell cx , the entering event is en_{cx}^X . To simplify modeling specification, we only consider the entering events because the event of exiting one cell in fact means the event of entering the next adjacent cell. Thus, there exist the following events: $en_{c6}^J, en_B^J, en_{c11}^A, en_{c7}^A, en_{c3}^A, en_{c8}^B, en_{c7}^B, en_{c6}^B, en_{c5}^B$.

Note that if there exist two or more vehicles in the same cell, they will collide. To avoid collision, when the buses A and B enter the cell $c7$, they must be scheduled so that they pass through the cell $c7$ in sequence. Additionally, John should enter the bus B in the cell $c6$ before B leaves.

Since a time dependency structure can represent a real-time mobile system, it is used to denote such a real-time mobile system. A real-time mobile system \mathcal{TDS} may contains multiple mobile objects and ambients. For convenience, the notation $\mathcal{TDS}_x \sqsubset \mathcal{TDS}$ is used to denote that \mathcal{TDS}_x is a mobile object or ambient of a real-time mobile system \mathcal{TDS} .

III. SYSTEM MODEL

An event is a core concept here, which means an occurrence of an activity or action. If an event occurs, such an event is said to be *available*; otherwise it is *unavailable*. The dependency structure model [4], [5] uses an *event set* (a set of events) as a basic element. If all the events in an event set are *available*, such an event set is said to be *available*; otherwise it is said to be *unavailable*. We equip events and the relationship among events with time attributes, and introduce the *time dependency structure*.

Definition III.1 A *time dependency structure* (\mathcal{TDS}) is a tuple $\langle \mathcal{E}, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{W}, \mathbb{F}, \mathcal{T}_i, \mathcal{T}_e, \mathcal{T}_t \rangle$ with

- \mathcal{E} , a finite set of events,
- $\mathbb{I} \subseteq 2^{\mathcal{E}}$, the set of initially available event sets,
- $\mathbb{T} \subseteq 2^{\mathcal{E}} \setminus \{\emptyset\} \times 2^{\mathcal{E}} \setminus \{\emptyset\}$, the (asymmetric) *transformation* relation,
- $\mathbb{S} \subseteq 2^{\mathcal{E}}$, the *synchronism* relation such that $\forall A \in \mathbb{S} : |A| > 1$,
- $\mathbb{C} \subseteq 2^{\mathcal{E}}$, the *choice* relation such that $\forall A \in \mathbb{C} : |A| > 1$,
- $\mathbb{W} : \mathcal{E} \rightarrow \{1, 2, 3, \dots\}$, the capacity function,
- $\mathbb{F} \subseteq 2^{\mathcal{E}}$, the set of finally available event sets,

- $\mathcal{T}_i : \bigcup_{X \in \mathbb{I}} X \rightarrow \mathbf{Time}$, the initial time function,
- $\mathcal{T}_e : \mathcal{E} \rightarrow \mathbf{Time}$, the event delay function, and
- $\mathcal{T}_t : \mathbb{T} \rightarrow \mathbf{Time}$, the transformation delay function.

Here, for all $A, B \in 2^{\mathcal{E}}$, $(A, B) \in \mathbb{T}$ is called a *transformation dependency*, denoted as $A \rightarrow B$, all read as B depending on A , and A, B are called the *pre-* and *post-dependency set* of the dependency (A, B) , respectively. The events in A, B are called the *pre-* and *post-events* of (A, B) , respectively.

Transformation is a binary relation between event sets where a transformation dependency $(A, B) \in \mathbb{T}$ is that the occurrences of all the events in B depends on the occurrences of all the events in A . A set $A \in \mathbb{S}$ and a set $B \in \mathbb{C}$ are called a *synchronism set* and a *choice set*, respectively. The capacity function \mathbb{W} restricts the available number of events, that is, if an event e may cause the occurrence of n events, then the capacity of such an event is n ($\mathbb{W}(e) = n$). The capacity function is similar to the token capacity function of places in a Petri net [9], which is used to control a loop.

To support multiple clock modeling, the initial time function is introduced. $\mathcal{T}_i(e)$ refers to the initial clock valuation of the initial available event e . The occurrence of an event may go on for some time. $\mathcal{T}_e(e)$ specifies the timing constraint of the event e . A transformation dependency expresses the dependency relationship between the two event sets and may have a time delay constraint. If a transformation dependency (A, B) has time delay t' , $\mathcal{T}_t((A, B)) = t'$.

In our running example (see Figure 1(b)), we can assume that it takes 1 time unit to enter an ambient and takes 2 time units to cross a ambient for vehicle A and B . we assume that the initial time of the vehicles A, B and passenger John are 3, 0, and 2, respectively. Therefore, the running example can be modeled as $\mathcal{TDS}_{run} = \langle \mathcal{E}, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{W}, \mathbb{F}, \mathcal{T}_i, \mathcal{T}_e, \mathcal{T}_t \rangle$ where

$\mathcal{E} = \{en_{c11}^A, en_{c7}^A, en_{c3}^A, en_{c8}^B, en_{c7}^B, en_{c6}^B, en_{c5}^B, en_{c6}^J, en_B^J\}$,
 $\mathbb{I} = \{\{en_{c11}^A\}, \{en_{c8}^B\}, \{en_{c6}^J\}\}$,
 $\mathbb{T} = \{(\{en_{c11}^A\}, \{en_{c7}^A\}), (\{en_{c7}^A\}, \{en_{c3}^A\}), (\{en_{c8}^B\}, \{en_{c7}^B\}), (\{en_{c7}^B\}, \{en_{c6}^B\}), (\{en_{c6}^B, en_{c6}^J\}, \{en_B^J\}), (\{en_B^J\}, \{en_{c5}^B\})\}$,
 $\mathbb{S} = \{\{en_{c6}^B, en_{c6}^J\}\}$, $\mathbb{C} = \emptyset$, $\forall e \in \mathcal{E}, \mathbb{W}(e) = \infty$, $\mathbb{F} = \{\{en_{c3}^A\}, \{en_{c5}^B\}\}$, and the timing constraints are as follows:

$\mathcal{T}_i(en_{c11}^A) = 3, \mathcal{T}_i(en_{c8}^B) = 0, \mathcal{T}_i(en_{c6}^J) = 2$,
 $\mathcal{T}_e(en_{c11}^A) = \mathcal{T}_e(en_{c7}^A) = \mathcal{T}_e(en_{c3}^A) = \mathcal{T}_e(en_{c8}^B) = \mathcal{T}_e(en_{c7}^B) = \mathcal{T}_e(en_{c6}^B) = \mathcal{T}_e(en_{c5}^B) = \mathcal{T}_e(en_{c6}^J) = \mathcal{T}_e(en_B^J) = 1$,
 $\mathcal{T}_t(\{en_{c11}^A\}, \{en_{c7}^A\}) = \mathcal{T}_t(\{en_{c7}^A\}, \{en_{c3}^A\}) = \mathcal{T}_t(\{en_{c8}^B\}, \{en_{c7}^B\}) = \mathcal{T}_t(\{en_{c7}^B\}, \{en_{c6}^B\}) = \mathcal{T}_t(\{en_{c6}^B, en_{c6}^J\}, \{en_B^J\}) = \mathcal{T}_t(\{en_B^J\}, \{en_{c5}^B\}) = 2$.

A system can just run along the path formed by its transformation dependencies. Synchronism, choice and timing constraints only control the execution of such a system. While a system runs, each of transformation dependencies may lead to the change of its states. With the passage of time, a transformation dependency may be *activated*. Only activated transformation dependencies will be possibly executed. Thus, a state includes the current possibly available events, the number of possibly activating transformation dependencies, the “absolute” time of the occurrence of every possibly available event, and currently activated transformation dependencies.

Definition III.2 Let $\mathcal{TDS} = \langle \mathcal{E}, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{W}, \mathbb{F}, \mathcal{T}_i, \mathcal{T}_e, \mathcal{T}_t \rangle$ be a time dependency structure. A *state* of \mathcal{TDS} is a tuple $\mathcal{S} = \langle \Delta, F, f_t, \Gamma \rangle$ where $\Delta \subseteq \mathcal{E}$ is the set of possibly available events, the *availability function* $F : \Delta \rightarrow \mathbb{Z}^*$ is a function from Δ to the set \mathbb{Z}^* of nonnegative integers, the time function $f_t : \Delta \rightarrow \mathbf{Time}$, and $\Gamma \subseteq \mathbb{T}$ is the set of activated transformation dependencies satisfying for all dependencies $(A, B) \in \Gamma \Rightarrow A \subseteq \Delta$. The *initial state* of \mathcal{TDS} is defined as $\mathcal{S}_0 = \langle \Delta_0, F_0, f_{t0}, \Gamma_0 \rangle$ such that $\Delta_0 = \bigcup_{X \in \mathbb{I}} X, \forall e \in \Delta_0 : F_0(e) = |\{(A, B) \in \mathbb{T} \mid e \in A\}|$, $\forall e \in \Delta_0 : f_{t0}(e) = \mathcal{T}_i(e) + \mathcal{T}_e(e)$, and $\Gamma_0 = \{(A, B) \mid A \in \mathbb{I}, (A, B) \in \mathbb{T}\}$.

For example, the initial state of \mathcal{TDS}_{run} is $\mathcal{S}_0 = \langle \Delta_0, F_0, f_{t0}, \Gamma_0 \rangle$ where $\Delta_0 = \{\{en_{c11}^A\}, \{en_{c8}^B\}, \{en_{c6}^J\}\}$, $F_0(en_{c11}^A) = F_0(en_{c8}^B) = F_0(en_{c6}^J) = 1$, $f_{t0}(en_{c11}^A) = 4$, $f_{t0}(en_{c8}^B) = 1$, $f_{t0}(en_{c6}^J) = 3$ and $\Gamma_0 = \{(en_{c11}^A, en_{c7}^A), (en_{c8}^B, en_{c7}^B), (\{en_{c6}^J, en_{c6}^B\}, en_{c6}^J)\}$.

For convenience, the state $\langle \Delta, F, f_t \rangle$ is denoted as $\{\langle e, F(e), f_t(e) \rangle \mid e \in \Delta\}$. The availability function F is similar to the marking of Petri nets. Given an event e and $F(e) = n$, n is called the *availability value* of e .

Given a synchronism set C , the *latest available time delay* of the events in a synchronism set is denoted by $Max\{f_t(e) \mid e \in C\}$. Note that since the absolute time of the occurrence of the events in a synchronism set is computed from the initial state of a system, one cannot directly determine which events occur in what order before the system starts to run.

Definition III.3 Let $\mathcal{TDS} = \langle \mathcal{E}, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{W}, \mathbb{F}, \mathcal{T}_i, \mathcal{T}_e, \mathcal{T}_t \rangle$ be a time dependency structure and $\mathcal{S}_1 = \langle \Delta_1, F_1, f_{t1}, \Gamma_1 \rangle, \mathcal{S}_2 = \langle \Delta_2, F_2, f_{t2}, \Gamma_2 \rangle$ be two of its states.

\mathcal{S}_1 can evolve into \mathcal{S}_2 by executing a transformation dependency (A, B) , denoted by $\mathcal{S}_1 \xrightarrow{(A, B)} \mathcal{S}_2$, if the following conditions hold:

- (1) $(A, B) \in \Gamma_1$,
- (2) $\nexists (E, F) \in \Gamma_1 : Max\{f_{t1}(e) \mid e \in E\} + \mathcal{T}_t((E, F)) < Max\{f_{t1}(e) \mid e \in A\} + \mathcal{T}_t((A, B))$,
- (3) $\Delta_2 = \{e \in \Delta_1 \mid e \notin A \vee (F_1(e) - (1+x) > 0 \wedge e \in A)\} \cup B$,
- (4) $\forall e \in \Delta_2 : F_2(e) \leq \mathbb{W}(e) \wedge F_2(e) = \begin{cases} F_1(e) - (1+x) & : e \in A \setminus B \\ F_1(e) & : e \in (\Delta_1 \setminus (A \cup B)) \\ F_1(e) - (1+x) + y & : e \in A \cap B \\ F_1(e) + y & : e \in (\Delta_1 \setminus A) \cap B \\ y & : e \in B \setminus \Delta_1 \end{cases}$

where $y = |\{(X, Y) \in \mathbb{T} \mid X \cap B \neq \emptyset\}|$ and $x = |\{(A, X) \in \mathbb{T} \mid \exists e \in X, \exists e' \in B, \exists C \in \mathbb{C} : e \neq e' \wedge \{e, e'\} \in C\}|$,

(5) $\Gamma_2 = (\Gamma_1 \setminus (\{(A, B)\} \cup B^c)) \cup B^T \cup B^S$ where $B^T = \{(B, X) \mid (B, X) \in \mathbb{T}\}$, $B^S = \{(X, Y) \in \mathbb{T} \mid X \in \mathbb{S}, X \subseteq \Delta_1 \cup B, B \subseteq X, Y \subseteq \mathcal{E}\}$ and $B^c = \{(W, X) \in \mathbb{T} \mid W \subseteq \mathcal{E}, \exists e \in X, \exists e' \in B, \exists C \in \mathbb{C} : e \neq e' \wedge \{e, e'\} \in C\}$, and

(6) $\forall e \in \Delta_2 : f_{t2}(e) = \text{if } e \in B \text{ then } Max\{f_{t1}(e') \mid e' \in A\} + \mathcal{T}_t((A, B)) + \mathcal{T}_e(e) \text{ else } f_{t1}(e)$.

According to the condition (2) of the preceding definition, we have $\mathcal{S}_0 \xrightarrow{(en_{c8}^B, en_{c7}^B)} \mathcal{S}_1$ in the running example, and then we have $\Delta_1 = \{\{en_{c11}^A\}, \{en_{c7}^B\}, \{en_{c6}^J\}\}$ (by the condition (3)), $F_1(en_{c11}^A) = F_1(en_{c7}^B) = F_1(en_{c6}^J) = 1$ (by the condition (4)), and $\Gamma_1 = \{(en_{c11}^A, en_{c7}^A), (en_{c7}^B, en_{c6}^B), (\{en_{c6}^J, en_{c6}^B\}, en_{c6}^J)\}$ (by the condition (5)). Thus, $\mathcal{S}_1 = \langle \Delta_1, F_1, f_{t1}, \Gamma_1 \rangle$ where

$f_{t1}(en_{c11}^A) = 4$, $f_{t1}(en_{c7}^B) = 4$, and $f_{t1}(en_{c6}^J) = 3$ (by the condition (6)).

A time dependency structure can be used to reason about the behavior and properties of a real-time system. We define some properties of a time dependency structure here.

Definition III.4 Let $\mathcal{TDS} = \langle \mathcal{E}, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{W}, \mathbb{F}, \mathcal{T}_i, \mathcal{T}_e, \mathcal{T}_t \rangle$ be a time dependency structure and \mathcal{S}_0 be the initial state of \mathcal{TDS} . Let $\mathcal{S}, \mathcal{S}'$ be two states of \mathcal{TDS} . A state \mathcal{S} is said to be *reachable from* \mathcal{S}' , denoted as $\mathcal{S}' \xrightarrow{*} \mathcal{S}$, if there exist the states $\mathcal{S}'_1, \dots, \mathcal{S}'_{n-1}$ such that $\mathcal{S}' \xrightarrow{d'_1} \mathcal{S}'_1 \xrightarrow{d'_2} \dots \xrightarrow{d'_n} \mathcal{S}$ ($d'_i \in \mathbb{T}, i \in \{1, \dots, n\}$). $Sta(\mathcal{TDS})$ denotes the set of all reachable states in \mathcal{TDS} .

IV. SCHEDULING AND ISOLATION CONTROL

In the section, we will introduce the notion of a schedule and analyze the isolation relationship of mobile objects in a real-time mobile system in order to explore the isolation.

Definition IV.1 Let $\mathcal{TDS} = \langle \mathcal{E}, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{W}, \mathbb{F}, \mathcal{T}_i, \mathcal{T}_e, \mathcal{T}_t \rangle$ be a time dependency structure.

A sequence $f = \mathcal{S}_0 X_1 \mathcal{S}_1 \dots X_n \mathcal{S}_n$ is called a *full sequence* of \mathcal{TDS} iff $\mathcal{S}_0 = \langle \Delta, F, f_t, \Gamma \rangle, \mathcal{S}_1 = \langle \Delta_1, F_1, f_{t1}, \Gamma_1 \rangle, \dots, \mathcal{S}_n = \langle \Delta_n, F_n, f_{tn}, \Gamma_n \rangle$ are states in \mathcal{TDS} and $\forall i \in \{0, 1, \dots, n\}, X_i \subseteq \mathcal{E}$ such that $\mathcal{S}_0 \xrightarrow{*} \mathcal{S}_1 \xrightarrow{*} \dots \xrightarrow{*} \mathcal{S}_n$ and $\forall i \in \{1, \dots, n\}, \forall e_1, e_2 \in X_i : (X_i \cap \Delta_i = X_i) \wedge (f_{ti-1}(e_1) = f_{ti-1}(e_2))$. Here, the sequence $s = X_1 \dots X_n$ is called a *schedule* of \mathcal{TDS} or is said to be *schedulable* in \mathcal{TDS} . $Sches(\mathcal{TDS})$ denotes the set of all the schedules in \mathcal{TDS} .

As the Definition IV.1, $f_{run} = \mathcal{S}_0 \{en_{c8}^B\} \mathcal{S}_1 \{en_{c11}^A, en_{c7}^B\} \mathcal{S}_2 \{en_{c7}^A, en_{c6}^B, en_{c6}^J\} \mathcal{S}_3 \{en_{c6}^J\} \mathcal{S}_4$ is a full sequence of \mathcal{TDS}_{run} , and $s_{run} = \{en_{c8}^B\} \{en_{c11}^A, en_{c7}^B\} \{en_{c7}^A, en_{c6}^B, en_{c6}^J\} \{en_{c6}^J\}$ is a schedule of \mathcal{TDS}_{run} .

A schedule is an ordered event set sequence, where the events in the front event set occur prior to those in the back event set. The scheduler of a system in fact is a controller that restricts the behaviour of such a system so that given scheduling requirements are met [1].

Definition IV.2 Let \mathcal{TDS} be a time dependency structure and $s \in Sches(\mathcal{TDS})$. The *restriction of \mathcal{TDS} to the schedule s* is denoted by $\mathcal{TDS}|_s$.

In this definition, $\mathcal{TDS}|_s$ means the time dependency structure \mathcal{TDS} whose behavior is restricted to the schedule s or the time dependency structure \mathcal{TDS} runs in terms of the control of the schedule s .

Proposition IV.1 Let $\mathcal{TDS} = \langle \mathcal{E}, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{W}, \mathbb{F}, \mathcal{T}_i, \mathcal{T}_e, \mathcal{T}_t \rangle$ be a time dependency structure and let s be a schedule in \mathcal{TDS} , then $Sta(\mathcal{TDS}|_s) \subseteq Sta(\mathcal{TDS})$.

The proposition shows that the states of scheduled mobile system are part of those of the original system, respectively.

Definition IV.3 Let \mathcal{TDS} be a time dependency structure and $s = X_1 \dots X_n \in Sches(\mathcal{TDS})$. The *restriction of \mathbf{A} to the schedule s* is defined as $\mathbf{A}|_s = \{A \in \mathbf{A} \mid \exists \mathcal{M} \in \mathbf{M} : en_A^M \in X_1 \cup \dots \cup X_n\}$.

In fact we use $\mathbf{A}|_s$ to denote the set of all the ambients that are involved in the schedule s .

Definition IV.4 Let \mathcal{TDS} be a time dependency structure. Let $\mathbf{M}_s \subseteq \mathbf{M}$. Let $A \in \mathbf{A}, \mathcal{M}_1, \mathcal{M}_2 \in \mathbf{M}_s$ and $A, \mathcal{M}_1, \mathcal{M}_2 \sqsubset \mathcal{TDS}$.

TABLE I
STATES OF THE TIME DEPENDENCY STRUCTURE OF THE RUNNING EXAMPLE SYSTEM

Source state	$\langle \Delta, F, f_t \rangle$	Γ	ED	Target state
S_0	$\{ \langle en_{c11}^A, 1, 4 \rangle, \langle en_{c8}^B, 1, 1 \rangle, \langle en_{c6}^J, 1, 3 \rangle \}$	$\{ (\{en_{c11}^A\}, \{en_{c7}^A\}), (\{en_{c8}^B\}, \{en_{c7}^B\}) \}$	$(\{en_{c8}^B\}, \{en_{c7}^B\})$	S_1
S_1	$\{ \langle en_{c11}^A, 1, 4 \rangle, \langle en_{c7}^B, 1, 4 \rangle, \langle en_{c6}^J, 1, 3 \rangle \}$	$\{ (\{en_{c11}^A\}, \{en_{c7}^A\}), (\{en_{c7}^B\}, \{en_{c6}^B\}) \}$	$(\{en_{c11}^A\}, \{en_{c7}^A\}), (\{en_{c7}^B\}, \{en_{c6}^B\})$	S_2
S_2	$\{ \langle en_{c7}^A, 1, 7 \rangle, \langle en_{c6}^B, 1, 7 \rangle, \langle en_{c6}^J, 1, 3 \rangle \}$	$\{ (\{en_{c7}^A\}, \{en_{c3}^A\}), (\{en_{c6}^B, en_{c6}^J\}, \{en_{c7}^B\}) \}$	$(\{en_{c7}^A\}, \{en_{c3}^A\}), (\{en_{c6}^B, en_{c6}^J\}, \{en_{c7}^B\})$	S_3
S_3	$\{ \langle en_{c3}^A, 0, 10 \rangle, \langle en_{c5}^J, 1, 10 \rangle \}$	$\{ (\{en_{c5}^J\}, \{en_{c5}^B\}) \}$	$(\{en_{c5}^J\}, \{en_{c5}^B\})$	S_4
S_4	$\{ \langle en_{c3}^A, 0, 10 \rangle, \langle en_{c5}^B, 0, 13 \rangle \}$			

Note that "ED" means currently executed transformation dependencies.

\mathcal{M}_1 is said to be *isolated from* \mathcal{M}_2 for \mathcal{A} in \mathcal{TDS} , denoted by $\mathcal{M}_1 \circ_{\mathcal{A}} \mathcal{M}_2$ in \mathcal{TDS} , iff either $\forall s \in \text{Sches}(\mathcal{TDS}), \mathcal{A} \notin \mathbf{A} \uparrow_s$, or $\forall s = B_1 \cdots B_n \in \text{Sches}(\mathcal{TDS}), (\exists \mathcal{X} \in \mathbf{A}, \nexists en_{\mathcal{A}}^{\mathcal{M}_2} \in B_1 \cup \cdots \cup B_n : en_{\mathcal{A}}^{\mathcal{M}_1} \in B_1 \wedge en_{\mathcal{X}}^{\mathcal{M}_1} \in B_n) \vee (\exists \mathcal{Y} \in \mathbf{A}, \nexists en_{\mathcal{A}}^{\mathcal{M}_1} \in B_1 \cup \cdots \cup B_n : en_{\mathcal{A}}^{\mathcal{M}_2} \in B_1 \wedge en_{\mathcal{Y}}^{\mathcal{M}_2} \in B_n)$.

\mathbf{M}_s is the set of mobile objects which need to isolate in order to avoid collision. For example, in the running example, we let $\mathbf{M}_s = \{A, B\}$ because of vehicle A and B need to isolate while John as a passenger and the vehicle B are not isolated from each other in the ambient $c6$ so that John takes the vehicle B .

This definition shows that if \mathcal{M}_1 is isolated from \mathcal{M}_2 , this means one of the following three cases holds: (1) \mathcal{M}_1 and \mathcal{M}_2 do not enter \mathcal{A} , (2) one of \mathcal{M}_1 and \mathcal{M}_2 enters \mathcal{A} , and (3) when the two mobile objects \mathcal{M}_1 and \mathcal{M}_2 both need to enter \mathcal{A} , one does not enter the ambient \mathcal{A} until the other exits \mathcal{A} .

Theorem IV.1 Let $\mathcal{TDS} = \langle \mathcal{E}, \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathbf{C}, \mathbf{W}, \mathbf{F}, \mathbf{T}_i, \mathbf{T}_e, \mathbf{T}_t \rangle$ be a time dependency structure. Let $\mathbf{M}_s \subseteq \mathbf{M}$. Let $\mathcal{A} \in \mathbf{A}$, $\mathcal{M}_1, \mathcal{M}_2 \in \mathbf{M}_s$, $en_{\mathcal{A}}^{\mathcal{M}_1}, en_{\mathcal{A}}^{\mathcal{M}_2} \in \mathcal{E}$, and $\mathcal{A}, \mathcal{M}_1, \mathcal{M}_2 \sqsubset \mathcal{TDS}$. Let $s = X_1 X_2 \dots X_n \in \text{Sches}(\mathcal{TDS})$.

If $\forall s_1 = X_i X_{i+1} \dots X_j \in \text{Sches}(\mathcal{TDS}]_s, \exists x \in \mathbf{A} : x \neq \mathcal{A} \wedge en_{\mathcal{A}}^{\mathcal{M}_1} \in X_i \wedge en_{\mathcal{A}}^{\mathcal{M}_2} \in X_j \wedge en_{\mathcal{A}}^{\mathcal{M}_1} \in X_{i+1} \cup \dots \cup X_j$, then $\mathcal{M}_1 \circ_{\mathcal{A}} \mathcal{M}_2$ in $\mathcal{TDS}]_s$.

In the running example, because of $s_{run} = \{en_{c8}^B\} \{en_{c11}^A, en_{c7}^B\} \{en_{c7}^A, en_{c6}^B, en_{c6}^J\} \{en_{c7}^B\} \in \text{Sches}(\mathcal{TDS})$, we have $s_{run1} = \{en_{c11}^A, en_{c7}^B\} \{en_{c7}^A, en_{c6}^B, en_{c6}^J\} \in \text{Sches}(\mathcal{TDS}]_{s_{run}}$. Since $en_{c7}^B \in \{en_{c11}^A, en_{c7}^B\}$, $en_{c7}^A \in \{en_{c7}^A, en_{c6}^B, en_{c6}^J\}$, and $en_{c6}^B \in \{en_{c11}^A, en_{c7}^B, en_{c7}^A, en_{c6}^B, en_{c6}^J\}$, we have $A \circ_{c7} B$.

The theorem states that we can decide whether two mobile objects are isolated from each other for a single ambient under a given schedule.

Theorem IV.2 Let \mathcal{TDS} be a time dependency structure. Let $\mathbf{M}_s \subseteq \mathbf{M}$.

If $\forall \mathcal{M}, \mathcal{N} \in \mathbf{M}_s, \forall \mathcal{X} \in \mathbf{A}, \forall s \in \text{Sches}(\mathcal{TDS})$, \mathcal{M} is isolated from \mathcal{N} for \mathcal{X} in $\mathcal{TDS}]_s$, then $\forall \mathcal{M}', \mathcal{N}' \in \mathbf{M}_s, \forall \mathcal{X}' \in \mathbf{A} : \mathcal{M}' \circ_{\mathcal{X}'} \mathcal{N}'$ in \mathcal{TDS} .

Theorem IV.2 in fact shows that given the set of mobile objects and the set of ambients, we can decide whether multiple mobile objects are isolated from each other for multiple ambients under a given schedule by checking the

available movement events of the states in a real-time mobile system.

V. CONCLUSION

A time dependency structure has been introduced and discussed. Based on the time dependency structure model, we have presented an approach for modeling a real-time mobile system. We have also defined a schedule for the isolation of mobile objects and have investigated the isolation schedulability analysis in a real-time mobile system. These results may be used for intelligent transportation systems and autonomous driving systems. In the future, we will further explore the isolation control and scheduling policies of the concurrent complex real-time mobile system. In practice, we will develop the scheduling policy generation method and wish it to be really used for autonomous driving.

REFERENCES

- [1] Karine Altisen and et al. Scheduler modeling based on the controller synthesis paradigm. *Real-time Systems*, 23(1):55–84, 2002.
- [2] Neil C. Audsley and et al. Deadline monotonic scheduling theory and application. *Control Engineering Practice*, 1(1):71–78, February 1993.
- [3] Luca Cardelli and Andrew D Gordon. Mobile ambients. In *FoSSaCS*, pages 140–155. Springer, 1998.
- [4] Jian-Min Jiang and et al. Event-based mobility modeling and analysis. *TCPs*, 1(2):9:1–9:32, February 2017.
- [5] Jianmin Jiang and et al. Analyzing event-based scheduling in concurrent reactive systems. *ACM TECS*, 14(4):86:1–86:27, 2015.
- [6] Kyoung-Dae Kim and P. R. Kumar. Cyber-physical systems: A perspective at the centennial. *Proceedings of the IEEE*, 100(1):1287–1308, May 2012.
- [7] Qiao Li and R. Negi. Maximal scheduling in wireless ad hoc networks with hypergraph interference models. *IEEE Trans. Vehicular Technology*, 61(1):297–310, November 2012.
- [8] Chung Laung Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, November 1973.
- [9] Tadao Murata. Petri nets: properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
- [10] John A. Stankovic and et al. *Deadline scheduling for real-time systems-EDF and related algorithms*, volume 460. Springer, 1998.
- [11] Qinghui Tang and et al. A unified methodology for scheduling in distributed cyber-physical systems. *ACM TECS*, 11(S2), August 2012.
- [12] Fumin Zhang and et al. Task scheduling for control oriented requirements for cyber-physical systems. In *Proceedings of the RTSS*, pages 47–56. IEEE, 2008.

Combining VSM and BTM to Improve Requirements Trace Links Generation

Bangchao Wang, Rong Peng*, Zhuo Wang, Yaxin Zhao
School of Computer Science, Wuhan University, Wuhan 430072, China
E-mail: {wangbc, rongpeng, wz2017, 2016301500098}@whu.edu.cn

Abstract—Trace links between software artifacts provide available traceability information and in-depth insights for different stakeholders. Unfortunately, establishing trace links is a fallible, tedious, and labor-intensive task. To alleviate these problems, many Information Retrieval (IR) methods, such as Vector Space Model (VSM), Latent Semantic Indexing (LSI) and their variants, have been proposed to establish trace links automatically. In recent years, short-text artifacts (or even lack of documentation) become a new trend as more and more software systems are developed abiding by agile methodologies. It makes the effects of traditional IR-based trace links generation methods even worse. In this paper, Biterm Topic Model (BTM), which is good at dealing with short text, is introduced to solve the problem. A hybrid method combining VSM and BTM is proposed to generate requirements trace links. The empirical experiments conducted on three real and frequently-used datasets indicate that the hybrid method can achieve better performance, and the results can reach the “acceptable level” directly.

Keywords—requirements traceability, information retrieval, vector space model, biterm topic model, short-text artifacts

I. INTRODUCTION

Requirements traceability (RT) is defined as “the ability to describe and follow the life of a requirement, in both a forward and backward direction [1]. In other words, it can provide visibility of the required aspects of the software and system development process, which contributes to a better understanding of the software system under development [2]. Thus, RT is one of the most important Requirements Engineering (RE) activities. Practically, traceability information has been proven vital to a wide variety software engineering activities, such as requirements consistency checking [3], change impact analysis [4], software reuse [5], and verification and validation (V&V) [6].

In early research and practice, RT is often accomplished by linking requirements to various software artifacts (e.g., design documents, source codes, and test cases) manually through a requirements traceability matrix [7]. However, as software systems evolve over time, RT activities are always time-consuming, tedious, and fallible [8]. As described in our previous work, determining how to improve the automatic degree and efficiency of a consistent and effective tracing process is an important challenge for both academia and industry [9].

To overcome this challenge, Information Retrieval (IR) techniques have been introduced and become the most popular techniques in the area of trace links generation [9]. A typical process of generating trace links by IR-based techniques for natural language artifacts, generally consists of the following steps: document preprocessing, candidate link generating, analyzing and refining [5]. After preprocessing, trace links can be automatically established using IR-based models, such as Vector Space Model (VSM)

[10, 11, 12], Latent Semantic Indexing (LSI) [13, 14, 15], and Probability Model (PM) [16, 17, 18].

In IR-based requirements tracing methods, requirements documents and target artifacts are usually regarded as queries and documents, respectively. And these methods aim to match a query of keywords with a set of objects in the software repository and rank the retrieved objects based on how relevant they are to the query using a predefined similarity measure [19, 20]. The tenet underlying IR-based tracing methods is that artifacts having a high textual similarity probably share several concepts, so they are likely good candidates to be traced from one another [20, 21, 22]. This tenet bases on the assumption that consistent terminologies have been used throughout the project’s lifecycle.

However, as projects evolve, new and inconsistent terminologies gradually emerge into the systems, which declines the performance of retrieval engines [20]. Besides, short-text, low quality text, and different expression preferences in different artifacts also negatively affect the performance of IR-based methods. To deal with all the problems, many strategies have been proposed to improve IR-based tracing methods [9, 23]. In an empirical study [24], the statistically analysis result shows several widely used IR-based tracing methods, which include VSM, LSI, Jensen-Shannon (JS), are almost equivalent. Meanwhile, another study [25] also draws an important conclusion that Latent Dirichlet Allocation (LDA) is able to capture some important information missed by other exploited IR methods, while its performance is lowest. These two conclusions indicate the opportunity to improve performance through combining different techniques, such as VSM and LDA, VSM and Relational Topic Model (RTM) [26]. However, two shortcomings still impede the combination techniques to get good performance on short-text artifacts tracing: firstly, they severely suffer from the severe data sparsity problem [27]; secondly, they cannot draw good results without enough learning corpus.

The empirical results introduced above motivates our work. A novel topic model which is good at dealing with short texts and lack of learning corpus problem.

In this paper, (1) Biterm Topic Model (BTM), good at dealing with short texts and lack of learning corpus, is introduced to generate trace links for the first time; and (2) a good-effect combination way—collection “union” operation (\cup)—is proposed to combine VSM and BTM to constitute candidate links set.

The remainder of this study is organized as follows. Section II provides background information, Section III reports the process of our approach, and Section IV presents the details of experiment. Experiment results are presented and discussed in Section V. Finally, in Section VI, the conclusions and future work are discussed.

*Corresponding author.

II. BACKGROUND

A. Starting Point for Discussion

How to evaluate the quality and usability of RT techniques is vital to propose new automatic RT techniques. The paper [30] proposed an acceptable and practical one according to their industrial practices, as shown in Table I. The standard also shows that candidate link lists with high recall and low precision are preferable to candidate link lists with high precision and low recall [30], as human analysts are much better in examining a given link list and determining whether it belongs to the answer set than they are in detecting whether the current set of links is sufficient.

TABLE I. STANDARDS FROM HAYES^[30]

Measure	Acceptable	Good	Excellent
Recall	60% — 69%	70% — 79%	80% — 100%
Precision	20% — 29%	30% — 49%	50% — 100%
Lag	3 — 4	2 — 3	0 — 2

The scalability of methods is another widely concerned issue. In other word, requirements tracing methods are expected to be able to achieve high quality for both “small” and “large” datasets. According to [30], a “small” dataset consists of 3000 combinatorial links or less. For example, a dataset consisting of 27 use cases and 87 test cases would have $27 \times 87 = 2349$ combinatorial links. Conversely, any dataset with more than 3000 combinatorial links is considered large.

In this work, we refer the quality evaluation standard of requirements tracing methods and the size boundary of dataset presented by [30], which is a starting point for discussion with researchers and practitioners.

B. Related works

In LDA, each document has a corresponding multinomial distribution over T topics and each topic has a corresponding multinomial distribution over the set of words in the vocabulary of the corpus. In [25], LDA has been introduced to generate trace links and capture some important information missed by other exploited IR methods, while its performance is low — the precision is less than 0.1. In [26], another topic model named RTM has been used to requirements traceability recovery. RTM is established with a foundation on LDA. Specifically, the process of modeling document-words distribution is identical to the LDA generative process. In their work, they propose a two-steps approach to combine similarity scores computed by two different IR methods for trace links generation. Firstly, the similarity scores of the two methods are mapped to a standard normal distribution. Secondly, the normalized scores are combined through a weighted sum. The value of confidence parameters for two IR methods need to be determined by users based on their experience. Since the homologous of their principle, two common shortcomings impede they generate trace links.

On the one hand, both LDA and RTM reveal the latent topics within the text corpus by implicitly capturing the document-level word co-occurrence patterns [28]. Since a lot of software artifacts are short texts, directly applying either LDA or RTM on this kind of artifacts will suffer from the severe data sparsity problem[29]. On the other hand, lacking

enough learning corpus may be another threat to impede the wide use of LDA and RTM. It is because insufficient learning corpus may not lead to good results in two different modeling processes.

Compared to these related works, our work introduce a new topic model named BTM to generate trace links for the first time. This model solve the severe data sparsity problem on short texts and the bad learning results problem on insufficient learning corpus. Besides, obtaining union (\cup) is proposed as a good-effect way to improve the candidate links set. The details of our approach will be presented in the next section.

III. OUR APPROACH

In this section, the overall process of our approach and the details of combining VSM and BTM to generate trace links will be presented.

A. Artifacts feature analysis

Generally, there are two kinds of text artifacts:

(1) Short text artifacts. This kind of artifacts only contain several words, such as “One sentence” text artifacts. For example, the requirement 103 in EBT only has one sentence: “A user shall register as a subscriber”.

(2) Long text artifacts. This kind of artifacts contains abundant contents to be traced on, such as structural text artifacts. For example, an artifact “use case” always includes use case name, summary, and description; and an artifact “test case” usually consists of test case name and pre/postconditions, as presented in TABLE II and III.

TABLE II. AN EXAMPLE OF USE CASE FROM EASYCLINIC

<i>UC01</i>	
Use case name:	input registry laboratory
Summary:	The Operator has been recognized by system and has all the data that characterize the the registry of the laboratory. The data in the S I O not be modified Success. The registry of the laboratory is properly inserted inside the S I O .
Description:	1.View the mask to enter information needed 2.Inserts data about the registry of laboratory 3.Confirm placement 4.Verify the data entered 5.Stores data 6.Notify operation it is finished with success

TABLE III. AN EXAMPLE OF TEST CASE FROM EBT

<i>TC141</i>	
Test case name:	Establish Trace (2.1.1)(2.2.1)
Preconditions& Postconditions:	Preconditions: Subscriber is registered Steps Subscriber establishes a trace between a UML artifact and a requirement. Postconditions: A trace is established between the UML artifact and the requirement.

As VSM is good at analyzing long text artifacts and BTM is good at analyzing short artifacts, different preprocessing processes are adopted according to their features.

B. BTM-based tracing method

As described in Section I, directly applying conventional topic models (e.g. LDA and RTM) on short texts do not work well as these models implicitly capture the document-level word co-occurrence patterns to reveal topics, and thus suffer from the severe data sparsity and lacking enough

learning corpus in short documents [28]. Thus, uncovering the topics within short-text artifacts and their relevance becomes a new challenge.

In [28], Yan propose a novel method for modeling topics in short texts, referred as biterm topic model (BTM). The key idea of BTM is to learn topics over short texts based on the aggregated biterms, namely word pairs, in the whole corpus to tackle the sparsity problem in single document. In other word, any two distinct words in an artifact are firstly extracted as a biterm. For example, in the short-text use case “The user can change password.”, if the stop words “The” and “can” are ignored after preprocessing, there are three biterms, i.e. “user change”, “user password”, and “change password”.

As presented in [28], suppose α and β are the Dirichlet priors. The specific generative process of the corpus in BTM can be described as follows:

1. For each topic z
 - (a) draw a topic-specific word distribution $\phi_z \sim \text{Dir}(\beta)$
2. Draw a topic distribution $\theta \sim \text{Dir}(\alpha)$ for the whole collection
3. For each biterm b in the biterm set B
 - (a) draw a topic assignment $z \sim \text{Multi}(\theta)$
 - (b) draw two words: $w_i, w_j \sim \text{Multi}(\phi_z)$

Figure 1 shows the biterm topic model, where θ is the topic probability distribution in the BTM corpus; ϕ is the topic-word pair probability distribution; z is the serial number of topic for corresponding word pair, T is the number of topics; w_i, w_j are two words in the biterm; B is the number of word pairs in the entire corpus; all the word pairs shares the same topic distribution in corpus, each topic corresponds to a polynomial distribution of several word pairs and this Multinomial distribution is recorded as Φ , each word pair corresponds to a topic; α and β are the hyperparameters of Dirichlet prior distribution.

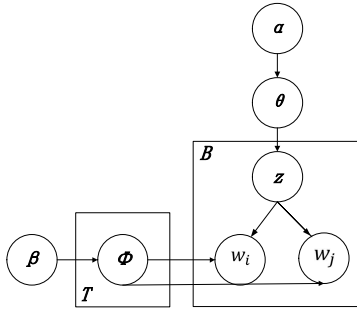


Fig. 1. Biterm topic model

As shown in Fig.2, the BTM-based trace links generation process can be divided into the following phases:

(1) Preprocessing. The preprocessing process is divided into two kinds: for short text artifacts, the pre-processing process contains stop words removal, part of speech tagging and word stemming; for long text artifacts, the upper pre-processing process is only executed on the most representative part of the artifact. For example, only the use/test case name is retained for the whole use/test case described in Table II/III.

(2) Biterms Extracting. In this step, any two distinct words got from the preprocessed source and target artifacts (namely the document collection) are firstly extracted as a biterm. And these biterms are used as the preprocessed corpus to train BTM.

(3) Biterm topic modeling. To use the preprocessed corpus training BTM, three parameters, including T , α and β , need to be set according to the experience related to the dataset. The outputs of this phase are: 1) topic distribution; 2) topic-word pairs distribution; and 3) document-topic distribution.

(4) Text similarity calculating. In this step, document-topic vectors must be established firstly. Based on these vectors, JS distance is used to represented the relevance between source and target artifacts. Note, it is different from cosine similarity since two texts are more similar when JS distance is smaller. After that, links are ranked according to JS distance.

(5) Candidate links generation. The generated links with top N similarity scores are selected as the candidate links.

Due to space limitations, we will not repeat the content about the specific probability calculation formulas of step (3) and JS distance calculation formulas of step (4). The details can be found in [28], where the BTM is proposed for the first time.

C. VSM-based tracing method

As shown in Fig.2, the VSM-based tracing method used in our approach can be divided into the following phases:

(1) Preprocessing. Both short artifacts and long artifacts are directly performed typical pre-processing steps: stop words removal, part of speech tagging and word stemming.

(2) Documents' vectors generation. A document d in the document collection is represented as a vector of keyword weights $d = (w_1, w_2, \dots, w_N)$, and the vocabulary of the entire collection is represented as (v_1, v_2, \dots, v_N) . The weight w_1 is calculated as the product of term frequency-inverse document frequency model (TF-IDF). Similarly, the query Q is also converted into a vector, represented as $q = (q_1, q_2, \dots, q_N)$.

(3) Text similarity calculation. And then, the relevance between document D and query Q is computed as the cosine of the angle between the vectors d and q , as represented in formula (1).

$$\text{sim}(d, q) = \cos(d, q) = \frac{\sum_{i=1}^N w_i \cdot q_i}{\sqrt{\sum_{i=1}^N w_i^2 \cdot \sum_{i=1}^N q_i^2}} \quad (1)$$

(4) Candidate links generation. After computing the cosine similarity, the retrieved objects are ranked based on the similarity scores. The artifacts with high textual similarity are likely good candidates to be traced from one another. In our approach, taking top N is adopt to determine the candidate links. In other words, the top N trace links with the highest scores will be selected as candidate links.

D. The hybrid method

Since complementarity exists between the VSM and BTM, “union” operation (\cup) has a great chance to improve the effect on combinations of precision and recall. The basic idea behind the hybrid method is that two IR methods, VSM and BTM, can be viewed as two experts who provide their expertise to generate candidate trace links. Both experts express judgments based on the textual similarity between two artifacts. The specific method is as following:

Suppose 1) the space size of a dataset is N ; 2) the dataset consisting of L true links and the count of the final candidate

links is nL ($n < N/L$); 2) the select ratios of VSM and BTM are λ and $1-\lambda$, respectively. Then, top $\lambda \times nL$ candidate links generated by VSM and top $(1-\lambda) \times nL$ candidate links generated by BTM are selected to construct the final trace links set. Note, after duplicate links removal, the count of the final trace links may be less than nL as some of the candidate links generated by VSM and BTM are same.

Finally, the candidate links generated in our hybrid method will be provided to analysts to obtain the final trace links as shown in Figure 2.

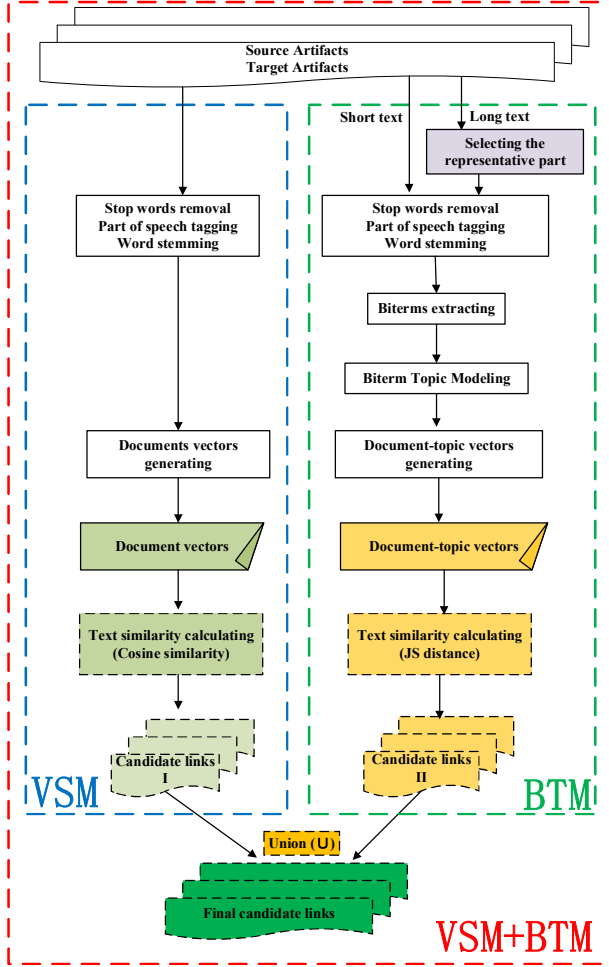


Fig. 2. The hybrid trace links generation method

IV. Experiment

This section describes the empirical experiments carried out on three real and frequently-used datasets to indicate that (1) it is reasonable to use different preprocess steps for VSM and BTM, (2) the hybrid method outperforms standard stand-alone IR methods, namely VSM and BTM.

It is note that the reason why LSI, JS and other similar methods are not regarded as comparison targets in our experiments is that VSM, LSI, JS and some similar methods are almost equivalent [24].

A. Research Question

To identify rationality and usefulness of the proposed hybrid method, the following two research questions need to be answered:

RQ1: Is it reasonable to use different preprocess steps for VSM and BTM?

RQ2: Does the hybrid method improve the quality of trace links generation and to what extent can this method achieve?

B. Datasets

Three datasets, WARC, EasyClinic, and EBT, are used to conduct the experiments. They are able to be downloaded at: <http://www.coest.org/>. The datasets are listed as following:

(1)WARC: This dataset includes 43 functional requirements (FRS), 21 non-functional requirements (NFR), 89 software requirements specification (SRS).

(2)EasyClinic: It is a small student-created dataset in English and Italian which contains diverse artifacts, including 30 Use Cases (UCs), 63 Test Cases (TCs), 20 Interaction Diagrams (IDs) and 47 Code Class descriptions (CCs). Note, only a subset of EasyClinic is chosen to conduct the experiment, as shown in Table III.

(3)EBT: It is an Event-Based Traceability (EBT) system, which contains 41 Requirements (Rqs), 25 Test Cases (TCs) and 52 Code Classes (CCs). Similarly, a subset of EBT is chosen to conduct the experiment, as shown in Table III.

Table IV shows the characteristics of each dataset used in the following experiments. Aiming to improve the availability of our method, the values of the parameters used in experiments are listed in Table V.

TABLE IV. DATASETS USED IN EXPERIMENTS

Dataset	Source	Target	Space (N)	True links (L)	Scale
WARC	43 FRS	89 SRS	3827	78	large
	21 NFR	89 SRS	1869	58	small
EasyClinic subset	30 UCs	63 TCs	1890	63	small
EBT subset	41 Rqs	25 TCs	1025	51	small

Note: "Space" represents the maximum counts of trace links.

TABLE V. PARAMETERS USED IN EXPERIMENTS

Dataset	T	α	β
WARC (FRS-SRS)	35	1.35	0.01
WARC (NFR-SRS)	35	1.35	0.01
EasyClinic (UC-TC)	20	1.25	0.01
EBT (Requirements-TC)	20	1.25	0.01

Note: $\lambda = 0.7$ is assigned to generate final trace links.

C. Quality Measures

Precision (P) and Recall (R) are the standard IR metrics to assess the quality of different requirements tracing techniques. Precision measures accuracy and recall measures coverage [17]. The combinations of precision and recall are used to evaluate the requirements tracing methods based on Hayes' standard described in Section II.

V. RESULTS AND DISCUSSION

In the following section, the findings of the experiments are clarified. After that, the potential threats are discussed.

A. Results

Before discussing and analyzing the results for **RQ1** and **RQ2**, some marks shown in Figure 3, 4, 5, 6 and 7 are illustrated.

As shown in Table I, the "Acceptable", "Good", and "Excellent" results must have a recall rate more than 60% and a precision rate no less than 20%. Therefore, in Figure 3

to 7, three areas, distinguished by green, blue, and pink dotted box, in the upper right corner of the coordinate axis are used to represent these three types of results, which are superimposed as “effective areas”.

For **RQ1**, experiments have been just performed in EasyClinic and EBT, as the WARC has no long-text artifacts. As Figure 3 shows, the quality of VSM on “long-text” (the preprocessing step is following the first step described in Section 3C) is obviously better than the quality of VSM on short-text ((the preprocessing step is following the first step described in Section 3B) in effective areas. However, BTM is better at dealing with short-text, as shown in [28]. Thus, it is reasonable to use different preprocess steps for VSM and BTM.

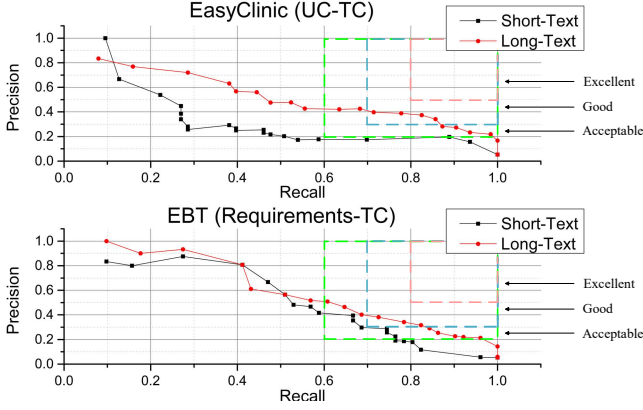


Fig. 3. The precision/recall distributions of VSM in short and long-text.

For **RQ2**, as shown in Figure 4, 5, and 6, all the red curves that fall in the effective area exceed the blue and black curves. In Figure 7, although both red and black curves can fall in the “Good” area, the quality of the red curve is better than the black one, which shows the combined method achieve the highest recall in the effective area.

Through the above analysis, the following conclusions can be summarized:

- (1) The hybrid method outperforms VSM and BTM in all three datasets. In other words, the hybrid method improves the quality of trace links generation;
- (2) The results of the hybrid method achieve “Good” level in EasyClinic, EBT and WARC (FRS-SRS) and “Acceptable” level in WARC (NFR-SRS).

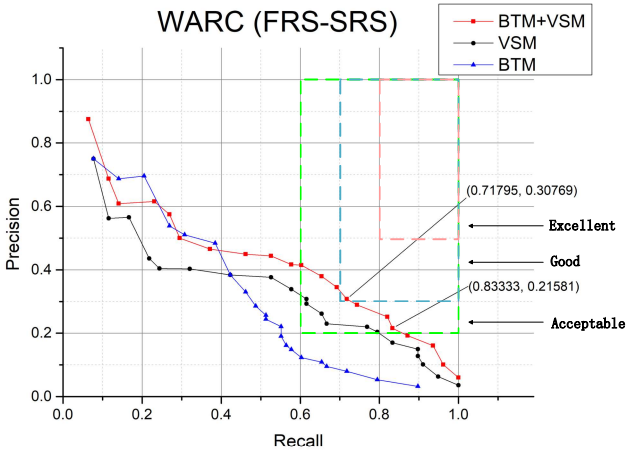


Fig. 4. The precision/recall curves of hybrid method and the standard VSM, BTM method in WARC (FRS-SRS).

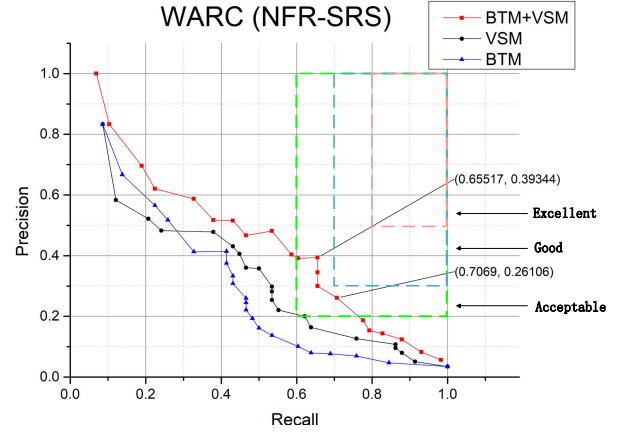


Fig. 5. The precision/recall curves of hybrid method and the standard VSM, BTM method in WARC (NFR-SRS).

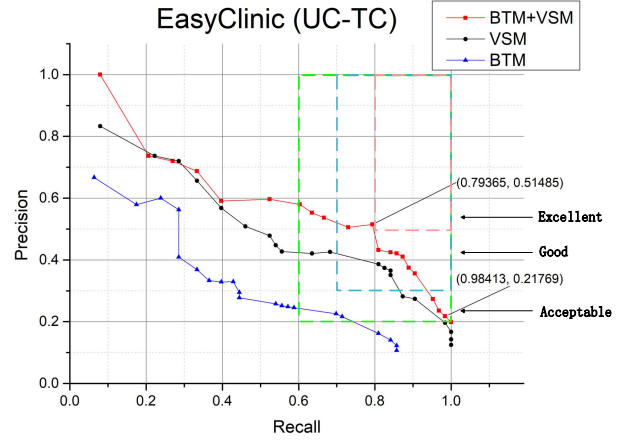


Fig. 6. The precision/recall curves of hybrid method and the standard VSM, BTM method in EasyClinic subset (UC-TC).

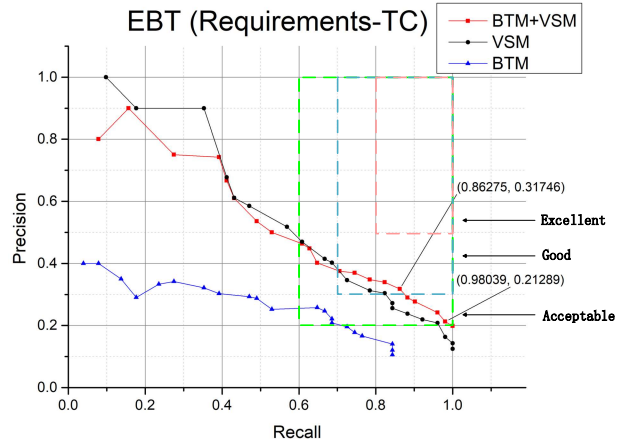


Fig. 7. The precision/recall curves of hybrid method and the standard VSM, BTM method in EBT subset (Requirements-TC).

B. Validity threats

This section aims to discuss the potential threats that influence the findings of this work. The following two aspects are discussed in this part.

Construct validity: (1) Since complementarity exists between the VSM and BTM, “union” operation (\cup) has a great chance to improve the effect. Therefore, it is reasonable to use these two methods to construct a hybrid tracing method. (2) As the result of RQ1 shows, VSM is good at analyzing long text artifacts and BTM is good at analyzing short artifacts, different preprocessing processes are adopted

according to their features. It ensures better effect can be obtained by the hybrid method. (3) The metrics used in our evaluation are recall and precision, which have been widely adopted for assessing the traceability accuracy of IR methods. Thus, we believe that they can sufficiently quantify the accuracy of three compared methods.

Conclusion validity: (1) In this work, experiment has been conducted on three real and frequently-used datasets. And, sufficient results and findings are summarized to illustrate the improvement. It illustrates the validity of our conclusions. (2) As the experiment results show, the hybrid method is able to achieve “Good” effect on small datasets as well as large datasets. The proposed hybrid method has a great chance to be generalized and introduced to other software systems and projects with various requirements and subsequent artifacts. (3) The validity of our experiment can also be affected by the chosen value of the parameter T , α , and β in the employed biterm topic model. We choose the value of three parameters based on our empirical evidence. In the future, we will obtain the optimal values by using some advanced techniques, such as optimization and machine learning.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, Biterm Topic Model (BTM), which is good at dealing with short texts and insufficient corpus, is introduced to generate trace links for the first time. After that, BTM is then combined with VSM to improve the effect of the standard IR methods. The experiments conducted on three real and frequently-used datasets in both large and small scale indicate that our method outperforms standard stand-alone IR methods, namely VSM and BTM.

In the future, we plan to perform detailed analysis about why BTM can complement some trace links for VSM-based tracing method. And then, some advanced parameter configuration techniques will be proposed to reduce the difficulty of using our method. Besides, all kinds of specific applicable scenarios for our method will be presented to improve the availability. Moreover, some refining strategies may be introduced to further improve the quality level.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Plan of China under Grant No. 2017YFB0503702, 2016YFB0501801, National Natural Science Foundation of China under Grant No. 61170026.

REFERENCES

- [1] Gotel O, Finkelstein A (1994) An analysis of the requirements traceability problem. In: International conference on requirements engineering, pp 94–101
- [2] Gotel, O., Cleland-Huang, J., Hayes, J.H., Zisman, A., Egyed, A., Gru'nbacher, P., Dekhtyar, A., Antoniol, G., Maletic, J., M'ader, P., 2012. Traceability fundamentals. In: Software and Systems Traceability. Springer, pp. 3–22
- [3] Sultanov H, Hayes J H, Kong W K. Application of swarm techniques to requirements tracing[J]. Requirements Engineering, 2011, 16(3):209-226.
- [4] von Knethen A (2002) Automatic change support based on a trace model. In: International workshop on traceability in emerging forms of software engineering
- [5] Spanoudakis G, Zisman A (2004) Software traceability: a roadmap. Handb Softw Eng Knowl Eng 3:395–428
- [6] Huffman-Hayes J, Dekhtyar A, Sundaram S (2006) Advancing candidate link generation for requirements tracing: the study of methods. IEEE Trans Softw Eng 32(1):4–19
- [7] Ramesh B, Jarke M (2001) Towards reference models for requirements traceability. IEEE Trans Softw Eng 27(1):58–93
- [8] Dekhtyar A, Huffman-Hayes J, Antoniol G (2007) Benchmarks for traceability? In: International symposium on grand challenges in traceability
- [9] Bangchao Wang, Rong Peng, Yuanbang Li, Han Lai, Zhuo Wang, Requirements traceability technologies and technology transfer decision support: A systematic review, The Journal of Systems & Software, 146C (2018) pp.59-79.
- [10] Niu, N., & Mahmoud, A. (2012). Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited. Requirements Engineering Conference (RE), 2012 20th IEEE International. IEEE.
- [11] Kong, W. K., & Hayes, J. H. (2011). Proximity-based traceability: An empirical validation using ranked retrieval and set-based measures. International Workshop on Empirical Requirements Engineering. IEEE.
- [12] Mahmoud, A., & Niu, N. (2013). Supporting requirements traceability through refactoring. Requirements Engineering Conference. IEEE.
- [13] Lormans, M., & Van Deursen, A. (2006). Can LSI help reconstructing requirements traceability in design and test?. Conference on Software Maintenance & Reengineering. IEEE.
- [14] Mcmillan, C., Poshyvanyk, D., & Revelle, M. (2009). Combining textual and structural analysis of software artifacts for traceability link recovery. Workshop on Traceability in Emerging Forms of Software Engineering. IEEE.
- [15] Wang, X., Lai, G., & Liu, C. (2009). Recovering Relationships between Documentation and Source Code based on the Characteristics of Software Engineering. Elsevier Science Publishers B. V.
- [16] Zou, X., Settimi, R., & Cleland-Huang, J. (2010). Improving automated requirements trace retrieval: a study of term-based enhancement methods. Empirical Software Engineering, 15(2), 119–146.
- [17] Zou, X., Settimi, R., & Clelandhuang, J. (2006). Phrasing in Dynamic Requirements Trace Retrieval. International Computer Software & Applications Conference. IEEE Computer Society.
- [18] Zou, X., Settimi, R., & Cleland-Huang, J. (2008). Evaluating the Use of Project Glossaries in Automated Trace Retrieval. International Conference on Software Engineering Research & Practice. DBLP.
- [19] Grzywaczewski A, Iqbal R (2012) Task-specific information retrieval systems for software engineers. J Comput Syst Sci 78(4):1204–1218
- [20] Mahmoud, A., & Niu, N. (2015). On the role of semantics in automated requirements tracing. Requirements Engineering, 20(3), 281–300.
- [21] Antoniol G, Caprile B, Potrich A, Tonella P (2000) Design-code traceability for object-oriented systems. Ann Softw Eng 9(1–4):35–58
- [22] De Lucia A, Fasano F, Oliveto R, Tortora G (2007) Recovering traceability links in software artifact management systems using information retrieval methods. ACM Trans Softw Eng Methodol 16(4):13–50
- [23] Hu Chenghai, Peng Rong, Wang Bangchao. A survey of requirement tracking method based on information retrieval. Computer Applications and Software, 2017(10):26-34.
- [24] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, “On the equivalence of information retrieval methods for automated traceability link recovery,” in Proc. of ICPC, 2010.
- [25] H. U. Asuncion, A. Asuncion, and R. N. Taylor, “Software traceability with topic modeling,” in Proc. of ICSE, 2010.
- [26] Gethers, Malcom, et al. "On integrating orthogonal information retrieval methods to improve traceability recovery," The College of Williams and." IEEE International Conference on Software Maintenance IEEE, 2011.
- [27] L. Hong and B. Davison. Empirical study of topic modeling in twitter. In Proceedings of the First Workshop on Social Media Analytics, pages 80–88. ACM, 2010.
- [28] X. Yan, J. Guo, Y. Lan, and X. Cheng, “A biterm topic model for short texts,” in Proc. 22nd Int. Conf. World Wide Web, 2013, pp. 1445–1456.
- [29] J. Chang and D. M. Blei, “Hierarchical relational models for document networks,” Annals of Applied Statistics, 2010.
- [30] Hayes J H, Dekhtyar A, Sundaram S K. Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods.[M]. IEEE Press, 2006.
- [31] Wohlin, C., Runeson, P., H'ost, M., Ohlsson, M.C., Regnell, B., Wesslen, A. Experimentation in software engineering, Springer Science & Business Media, 2012.

Themis: a tool for validating ontologies through requirements

Alba Fernández-Izquierdo
Ontology Engineering Group
Universidad Politécnica de Madrid
albafernandez@fi.upm.es

Raúl García-Castro
Ontology Engineering Group
Universidad Politécnica de Madrid
rgarcia@fi.upm.es

Abstract—The validation of ontologies, whose aim is to check whether an ontology matches the conceptualization it is meant to specify, is a key activity for guaranteeing the quality of ontologies. This work is focused on the validation through requirements, with the aim of assuring, both the domain experts and ontology developers, that the ontologies they are building or using are complete regarding their needs. Inspired by software engineering testing processes, this work proposes a web-based tool called Themis, independent of any ontology development environment, for validating ontologies by means of the application of test expressions which, following lexico-syntactic patterns, represent the desired behaviour that will present an ontology if a requirement is satisfied.

I. INTRODUCTION

In software engineering it is inconceivable to deliver a software product without its pertinent tests which guarantee that it fulfills all its requirements. Besides, there are several approaches integrated into the software development process whose aim is to test the software. Unit testing [1], which validates that each unit of the software performs as designed, and behaviour-driven development [2], which focuses on the behaviour the software product is implementing, are examples of these approaches. Moreover, there are specific syntaxes, such as Gherkin,¹ which generate unambiguous specifications of software to automate the testing process.

However, in ontology engineering there is a lack of clearly defined testing processes in order to be able to ascertain whether an ontology satisfies its functional requirements [3], which state the particular knowledge that should be represented. Such ontological requirements used to be written in form of competency questions [4] or natural language sentences. The main issue when performing testing processes in the ontology engineering field is the ambiguity of the ontological requirements, which sometimes are difficult to formalize into tests and to translate into axioms. Therefore, inspired by software engineering and its specific syntax for the definition of tests, we propose Themis,² a tool which provides a set of test expressions based on lexico-syntactic patterns (LSPs) related to ontological requirements. These LSPs allows to relate different types of requirements with the axioms needed to implement them in an ontology, and such implementations are used by Themis to identify whether a requirement is satisfied.

Themis can be used by both domain experts and ontology developers to validate ontologies regarding their functional requirements. Other type of requirements, such as non-functional ones (e.g., “the ontology URIs must be in English”) are not considered in this work as they cannot be formalized into axioms and, therefore, they cannot be automatically checked.³ Moreover, the proposed tool allows to execute tests on multiple ontologies simultaneously, in order to check ontological commitments between them.

The paper is organized as follows. Section 2 presents the state of the art on ontology testing tools. Section 3 presents Themis and Section 4 describes a use case in which Themis was integrated into a research project. Finally, Section 5 presents the conclusions we obtained and gives an overview of future work.

II. STATE OF THE ART

Currently, there are several methodologies and tools that support executing tests on an ontology, in order to validate that the ontology satisfies the pertinent requirements.

Regarding methodologies for testing ontologies, Vrandečić and Gangemi [5] introduced the idea of testing ontologies by borrowing ideas from software engineering, proposing techniques such as testing with axioms and negations or formalizing competency questions. Another work presented by Peroni is SAMOD [6], an ontology development methodology that uses tests for validation. These two approaches are focused on methodological aspects but do not mention how to implement the tests or how to maintain traceability.

Another approach to implement testing is the one presented by Ren et al. [7]; in this work the authors use natural language processing to analyse competency questions written in controlled natural language from where they create competency question patterns that could be automatically tested in the ontology. Additionally, Keet and Lawrynowicz proposed a test-driven development of ontologies [8] in which the competency questions are formalized into axioms and added to the ontology if they are not present.

Concerning testing tools, the OntologyTest tool [9] allows a user to define and execute a set of tests to check the functional requirements of an ontology; these tests are stored in an XML file for future reuse. It is worth mentioning that

DOI reference number: 10.18293/SEKE2019-117

¹<https://docs.cucumber.io/gherkin>

²<http://themis.linkedata.es>

³For the sake of clarity, from now on we are going to refer to functional requirements simply as requirements.

Another work related to ontology testing tools is Scone,⁴ a tool for scenario-based ontology evaluation, which is based on Cucumber⁵ and uses controlled natural language to define ontology scenarios which create mock individuals. Additionally, Blomqvist et al. [10] presented an agile approach and tool available as an Eclipse plugin. This tool supports three types of test, namely: (1) verification example, (2) inference verification and (3) error provocation. The first two are concerned with verifying the correct implementation of a requirement and the third is intended to expose faults. All the tests are stored in a different OWL ontology with information about the requirement, e.g., type of test or expected output. By saving these test suites in an ontology it allows the user to reuse them and to maintain traceability between the requirements and all the associated information. However, these tools neither explains how to implement these tests nor how to use them.

Even if all these works proposed solutions for testing through requirements, several of these works do not allow the reuse of the tests, limiting the testing process only to a single ontology. Moreover, there is a lack of information about how to translate a requirement into a test.

III. THEMIS IN THE TESTING PROCESS

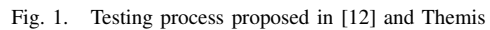
A. Test design

⁴<https://bitbucket.org/malefort/scone>

⁵<https://docs.cucumber.io>

⁶<https://protege.stanford.edu>

⁷<https://www.w3.org/TR/owl2-manchester-syntax>



Themis supports a list of possible tests expressions which are extracted from LSPs and that can be executed on an ontology. LSPs are understood as “formalized linguistic schemas or constructions derived from regular expressions in natural language that consist of certain linguistic and paralinguistic elements, following a specific syntactic order, and that permit to extract some conclusions about the meaning they express” [13]. The LSPs used by Themis were extracted from the CORAL corpus⁸ which, based on the NeOn modelling components [14], analyses 834 ontology requirements in order to identify LSPs based on the goal that each requirement has regarding its implementation in an ontology, e.g., a relation between two concepts. Therefore, these LSPs indicate the implementation in the ontology associated with a particular requirement template, which can be used to generate the corresponding tests.

In the Themis web user interface the user enters the test expression or set of test expressions that represent the desired behaviour of a requirement. Such translation between requirements and test expressions should be done manually

⁸<http://coralcorpus.linkeddata.es>

⁹<http://themis.linkeddata.es/examples.html>

TABLE I
TEST EXPRESSION CATALOGUE

Test goal	Test expression syntax	Test expression example	Example of requirement associated
T1 Equivalence	A EquivalentTo B	SecuritySchema equivalentTo Security	Security schema is equivalent to security
T2 Subsumption	A subClassOf B	Sensor subClassOf Device	A sensor is a type of device
T3 Disjointness	A disjointWith B	Sensor disjointWith Actuator	A sensor cannot be an actuator
T4 Property between two concepts	A subClassOf P some B	Bird subClassOf build some Nest	Birds build nests
T5 Universal restriction	A subClassOf P only B	User subClassOf interactsWith only Application	A user can only interact with the systems by means of an application
T6 Multiple inheritance	A subClassOf B and C	Sensor subClassOf System and Device	A sensor must be a system and a device
T7 Symmetry	A Symmetric(P) B	Partnership subClassOf symmetricProperty(hasPartnershipWith) some Organization	There is a partnership between two organizations
T8 Maximum cardinality	A subClassOf P max [num] B	Person subClassOf hasAddress max 1 Address	A person has at most 1 address
T9 Minimum cardinality	A subClassOf P min [num] B	Device subClassOf hasManufacturer min 1 string	A device has at least 1 manufacturer
T10 Cardinality	A subClassOf P max [num]B	Device subClassOf hasManufacturer exactly 1 Brand	A device has exactly 1 brand
T11 The ontology contains the individual	I type A	StriatedPardalote type Bird	A Striated Pardalote is an example of bird
T12 Subsumption and relation between classes	A subClassOf [ClassB] that [PropertyP] some C	Price subClassOf UnitOfMeasure that isCharacterizedBy some Currency	The price, which is a type of unit of measure, is characterized by a value using currency
T13 Minimum cardinality and relation between classes	A subClassOf [PropertyP] min [num]B and B subClassOf [PropertyP] some C	Device subClassOf performs min 1 Function and Function subClassOf accomplish some Task	A device performs at least 1 function and each function accomplishes some task
T14 Minimum cardinality and universal restriction	A subClassOf [PropertyP] min [num]B and B subClassOf [PropertyP] only C	Person subClassOf hasTask min 1 Task and Task subClassOf hasGoal only Goal	A person performs at least one task and each task has a goal
T15 Definition of a disjoint set of classes	A subClassOf B and C subClassOf B that disjointWith A	Carnivora subClassOf MarineMammals and Sirenia subClassOf MarineMammals that disjointWith Carnivora	Marine mammals are divided into two different types: Carnivora and Sirenia

using the guidelines provided in Themis website.¹⁰ As an example, the user can enter the test expression *Event subClassOf InteractionPattern* in order to check a subsumption relation in the ontology expected from the requirement “An event is a type of Interaction pattern”.

Additionally, Themis allows the users to export a set of test expressions, i.e., a test suite, as an RDF file in order to be able to reuse it. Listing 1 shows an example of the test expressions *Action subClassOf InteractionPattern* and *Event subClassOf InteractionPattern* in RDF syntax. Such test expressions aims to check subsumption relations in the ontology. This RDF file uses the ontology Verification Test Case¹¹ ontology to describe each test case.

This RDF file can be improved by adding the requirements associated to the tests, in order to provide a link between the test case and the requirements from which it is extracted. Listing 1 shows the test expressions *Action subClassOf InteractionPattern* and *Event subClassOf InteractionPattern* with the URIs of requirements associated, together with the description of the requirements which specifies what is an action and what is an event.

Listing 1. Example of improved test suite in RDF file

```
@prefix dc: <http://purl.org/dc/terms/> .
@prefix vtc: <http://w3id.org/def/vtc#> .
@prefix xsd: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix vicinity: <http://vicinity.iot.linkeddata.es/vicinity/requirements/report-wot.html#> .
@prefix : <#> .

: test-case-3 a vtc:TestCaseDesign;
  vtc:isRelatedToRequirement vicinity:wot16;
```

¹⁰<http://themis.linkeddata.es/tests-info.html>

¹¹<http://w3id.org/def/vtc>

```
dc:description "What is an action? The Action interaction pattern is an interaction pattern that targets changes or processes on a Thing that take a certain time to complete"
vtc:desiredBehaviour "Action subClassOf InteractionPattern"^^xsd:string .
```

```
: test-case-4 a vtc:TestCaseDesign;
  vtc:isRelatedToRequirement vicinity:wot17;
  dc:description "What is an event? The Event interaction pattern is an interaction pattern that enables a mechanism to be notified by a Thing on a certain condition.";
  vtc:desiredBehaviour "Event subClassOf InteractionPattern"^^xsd:string .
```

These RDF files can be also loaded in Themis, which will extract the test expressions that in the ontology are stored as *desiredBehaviour* data properties. The test suites can be modified and adapted to a particular ontology by the user. This functionality allows to reuse test suites already defined and published on the Web, e.g., test suites already defined for the ontology to be analysed, or test suites defined for other ontologies that could be used to check alignment.

B. Test implementation

During the test implementation activity each test expression is formalized into a precondition, a set of auxiliary term declarations and a set of assertions to check the behaviour. The precondition is a SPARQL query which checks whether the terms involved in the ontology requirement are defined in the ontology. The axioms to declare auxiliary terms are a set of temporary axioms added to the ontology to declare the auxiliary terms needed to carry out the assertions. Finally, the assertions to check the behaviour are a set of pairs of axioms and expected results that represent different ontology scenarios. For each pair, the axiom is temporary

added to the ontology to force a scenario, after which the reasoner is executed. The expected result determines if the ontology status (i.e., the ontology is inconsistent, there is an unsatisfiable class or the ontology is consistent) after the addition is the expected one in the case the requirement was satisfied. If all the status concurs with the expected status, then the requirement is satisfied. Figure 2 summarizes the flow of this test implementation.

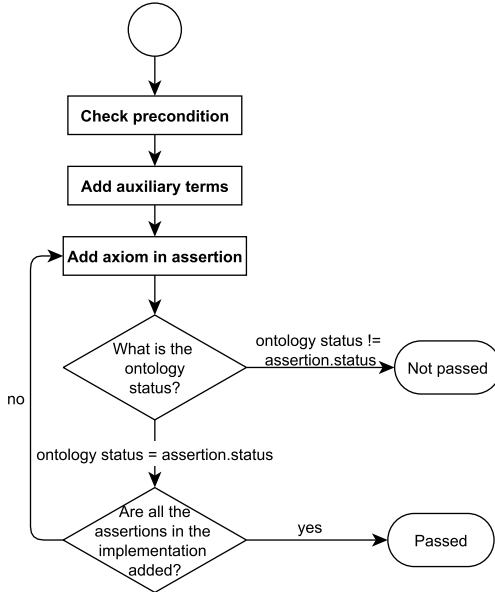


Fig. 2. Test implementation flow

Themis creates the implementation of each test expression listed in Table I. Once the user enters the test expression to be checked, Themis identifies the type of test and generates the correct implementation. As an example, if the test expression added by the user is *Action subClassOf InteractionPattern*, Themis will identify that it is of type *T2 Subsumption* and that the implementation includes¹²:

- **Preconditions:** Class Action and class InteractionPattern exist
- **Axioms to declare auxiliary terms:** Declaration of \neg Action and \neg InteractionPattern
- **Assertion 1:**
 - **Axiom:** Action' $\sqsubseteq \neg$ Action \sqcap InteractionPattern
 - **Expected status after adding the axiom:** Consistent
- **Assertion 2:**
 - **Axiom:** Action' \sqsubseteq Action $\sqcap \neg$ InteractionPattern
 - **Expected status after adding the axiom:** Unsatisfiable class
- **Assertion 3:**
 - **Axiom:** Action' \sqsubseteq Action \sqcap InteractionPattern
 - **Expected status after adding the axiom:** Consistent ontology

¹²This implementation is described by means of Description Logics symbols.

C. Test execution

Finally, during the test execution activity, the implementation of the test is executed in the ontology to be tested. At this point, the ontology URI needs to be indicated, as well as the URIs of each term in the test expression in order to be able to execute it, e.g., the terms Action and InteractionPattern in the ontology with URI <http://iot.linkeddata.es/def/wot#> will be <http://iot.linkeddata.es/def/wot#Action> and <http://iot.linkeddata.es/def/wot#InteractionPattern>, respectively. These terms URIs needs to be extracted from the glossary of terms of the analysed ontology.

Themis proposes a glossary of terms where the terms are extracted from the fragments of the URIs of each concept. Table II shows an excerpt of a glossary of terms created by Themis for the ontology VICINITY Web Of Things.¹³ This preliminary glossary of terms can be modified by the users if needed. Even though in this case the terms coincide with the URIs' fragments, that is not required.

TABLE II
EXCERPT OF GLOSSARY OF TERMS

Term	URI
Event	http://iot.linkeddata.es/def/wot#Event
Action	http://iot.linkeddata.es/def/wot#Action
InteractionPattern	http://iot.linkeddata.es/def/wot#InteractionPattern
CommunicationProtocol	http://iot.linkeddata.es/def/wot#CommunicationProtocol
isProvidedOverProtocol	http://iot.linkeddata.es/def/wot#isProvidedOverProtocol

The test execution activity consists of three parts: the execution of the query which represents the preconditions, the addition of the axioms which declare the auxiliary terms, and the addition of the assertions. After the addition of each axiom, the reasoner is executed to report the status of the ontology. The addition of the auxiliary axioms needs to always lead to a consistent ontology. In the case of the assertions, the agreement between the reasoner status after the addition of all the axioms and the status indicated in the test implementation determines whether the ontology satisfies the desired behaviour. All the results of each step are stored in RDF files, in order to enable traceability between them and the requirements.

Themis uses OWL API¹⁴ to load the ontologies and to add axioms, and uses Pellet¹⁵ as the reasoner to check the ontologies consistency. Themis can execute a test expression on several ontologies simultaneously, allowing the users to identify common knowledge and commitments between a collection of ontologies regarding their requirements.

Four possible results can be returned for each test and each ontology:

- 1) *Undefined terms:* if the ontology does not pass the preconditions, i.e., the terms in the test expression are not defined in the ontology to be analysed.
- 2) *Passed:* if the ontology passes the preconditions and the results of the assertions are the expected ones.

¹³<http://iot.linkeddata.es/def/wot>

¹⁴<http://owlapi.sourceforge.net/documentation.html>

¹⁵<https://www.w3.org/2001/sw/wiki/Pellet>

- 3) *Absent relation*: if the ontology passes the preconditions and the results of the assertion are not the expected ones but there are no conflicts in the ontology.
- 4) *Conflict*: if the ontology passes the preconditions and the results of the assertion are not the expected ones, and the addition of the axioms related to the test expression leads to a conflict in the ontology.

As an example, Themis can determine that the test expression *Action subclassOf InteractionPattern* is passed in the VICINITY Web Of Things ontology¹⁶ but has undefined terms in the oneM2M ontology¹⁷, due to the fact that the latter ontology does not consider the modelling of such concepts related to actions or interaction patterns.

IV. APPLICATION OF THEMIS

Themis has been used during the ontology development process of the VICINITY European h2020 project,¹⁸ where five ontologies are currently under development.

The five ontologies to be analysed in the project, i.e., the VICINITY Core (Core), the Web of Things (WoT), the WoT mappings (Mappings), the VICINITY Adapters (Adapters), and the Datatypes (Datatypes) ontologies, belong to the VICINITY ontology network and aim to provide interoperability in the IoT domain. The Core ontology represents the information needed to exchange IoT descriptor data between peers through the VICINITY platform; this ontology is being created by following a cross-domain approach and implements requirements from different domain experts. The WoT ontology aims to model the Web of Things domain according to the W3C WoT Interest Group¹⁹ descriptions. The Mappings ontology represents the mechanism for accessing the values provided by web things in the VICINITY platform. The Adapters ontology aims to model all the different types of devices and properties that can be defined in the VICINITY platform. Finally, the Datatypes ontology aims to model the required and provided datatypes that are used in the interaction patterns of the platform.

Table III summarizes the number of requirements defined for each ontology, as well as the number of test cases extracted from them. These requirements represent the needs asked by the domain experts in order to model the VICINITY platform. More information about these ontologies is available in the VICINITY ontology network portal.²⁰

The test cases were extracted by using the test expression catalogue provided in Table I. Each requirement is translated to one or more test expressions, selecting the most appropriate one from the test catalogue. Several test cases can be related to the same requirement. The test suite associated to each ontology, where all the test expressions are stored, was exported to an RDF file and uploaded to the VICINITY ontology portal,²¹ with the aim of reusing them in

future releases of the ontology to assure that all the previous requirements are still satisfied.

TABLE III
DETAILS OF THE ONTOLOGIES

Ontology	Number of requirements	Number of tests
Core ontology	50	50
WoT ontology	24	24
Mappings ontology	15	15
Adapters ontology	127	154
Datatypes ontology	11	12

Once all the tests were defined, Themis was executed in order to identify if there are tests that are not passed by the ontology. Figure IV summarizes the obtained results of such execution. As the table shows, the majority of the requirements are passed by the ontology. However, there are several tests whose results are *absent relation*, which means that the ontology passes the preconditions, the results of the assertion are not the expected ones but there are no conflicts in the ontology. These results warn the ontology developers that there are some tests that, even though they do not cause any conflict in the ontology, they are not implemented, at least completely, in the ontology.

TABLE IV
TESTING RESULTS OF THE VICINITY ONTOLOGY NETWORK

Ontology	Number of tests	Undefined terms	Passed	Absent relation	Conflict
Core	50	0	34	16	0
WoT	24	0	22	2	0
Mappings	15	0	13	2	0
Adapters	154	0	154	0	0
Datatypes	12	0	12	0	0

Additionally, the VICINITY ontology network should be aligned with the requirements of several standards in the IoT field in order to reuse concepts and patterns of well-known resources, namely: (1) the ETSI SAREF ontology²² [16], the W3C SSN ontology²³, (3) OCF standards²⁴, (4) the oneM2M ontology [16] and (5) ISO/IEC 30141:2017 [17]. Therefore, the requirements related to these standards were collected, and the associated tests were defined by using the test expression catalogue. Table V summarizes the number of requirements and test cases associated.

TABLE V
DETAILS OF THE EXTERNAL ONTOLOGIES

Ontology	Number of requirements	Number of tests
ETSI SAREF	70	70
W3C SSN	24	24
OCF	27	27
oneM2M	33	33
ISO/IEC 30141:2017	36	36

Themis was also used to check if the VICINITY ontology network satisfies the test expressions defined for the requirements of these ontologies and standards. From the execution

¹⁶<http://iot.linkeddata.es/def/wot>

¹⁷<https://git.onem2m.org/MAS/BaseOntology/raw/master/base.ontology.owl>

¹⁸<https://vicinity2020.eu/vicinity/>

¹⁹<http://w3c.github.io/wot/>

²⁰<http://vicinity.iot.linkeddata.es/>

²¹<http://vicinity.iot.linkeddata.es/vicinity/testing.html>

²²<https://w3id.org/saref>

²³<http://www.w3.org/ns/ssn/>

²⁴<https://openconnectivity.org/>

of Themis, it could be deduced that the VICINITY ontology network satisfies several tests, but did not take into consideration some concepts related to these ontologies and standards, as the developers did not consider it necessary for the project domain definition. However, it could also be concluded that there were no conflicts between the VICINITY ontology network and these ontologies and standards. Therefore, it can be determined that the VICINITY ontologies, even though they do not satisfy all the requirements related to the analysed IoT standards, are not incompatible with them. Table IV summarizes the results obtained after Themis execution. More information related to the results is available in the VICINITY ontology portal.²⁵

TABLE VI
TESTING RESULTS OF THE EXECUTION OF THE STANDARDS'
REQUIREMENTS IN VICINITY ONTOLOGY NETWORK

Standard	Number of tests	Undefined terms	Passed	Absent relation	Conflict
ETSI SAREF	70	66	4	0	0
W3C SSN	24	9	13	0	0
OCF	27	21	6	0	0
oneM2M	33	31	2	0	0
ISO/IEC 30141:2017	36	22	14	0	0

V. CONCLUSIONS AND FUTURE WORK

In this work we present Themis, a web-based tool to validate ontologies regarding their functional ontology requirements. To validate an ontology, Themis supports a set of different types of tests expressions, which are extracted from a collection of lexico-syntactic patterns with the aim of easing the formalization of requirements into tests cases. These lexico-syntactic patterns link requirements templates with possible implementations in an ontology and the test expressions check that these implementations are satisfied by a given ontology. Moreover, lexico-syntactic patterns use ontology design patterns to identify possible implementations. Therefore, Themis can also check if an ontology is satisfying a particular test following the ontology design patterns associated to the lexico-syntactic pattern.

Additionally, Themis allows to execute tests on a collection of ontologies simultaneously, which can be used to analyse the overlap of ontological commitment between them. This could be relevant if a developer requires to analyse whether a particular ontology is compliant with a given ontology or standard. The execution of tests on a collection of ontologies can also be used to ease the reuse of ontologies based on their requirements, due to the fact that the user is able to check which ontology satisfies the set of requirements he or she needs to cover.

As part of the continuous process of improving Themis, new tests can be added to the tool if new types of requirements or lexico-syntactic patterns are found. Furthermore, it is also planned to provide more types of results to the users, in addition to the four types that are provided so far.

This addition of possible results aims to help the users to detect more accurately which is the problem when a test is not passed and, therefore, to ease the repairing task. Future work will be also directed to the development of a REST service to be integrated in other ontology engineering tools, such as OnToolology.²⁶

VI. ACKNOWLEDGMENTS.

This work is partially supported by the H2020 project VICINITY: Open virtual neighbourhood network to connect intelligent buildings and smart objects (H2020-688467) and by a Predoctoral grant from the I+D+i program of the Universidad Politécnica de Madrid.

REFERENCES

- [1] P. Hamill, *Unit test frameworks: tools for high-quality software development*. O'Reilly Media, Inc., 2004.
- [2] M. Wynne, A. Hellesoy, and S. Tooke, *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf, 2017.
- [3] M. C. Suárez-Figueroa, A. Gómez-Pérez, and B. Villazón-Terrazas, "How to write and use the ontology requirements specification document," in *Proceedings of the International Conference on On the Move to Meaningful Internet Systems*, 2009, pp. 966–982.
- [4] M. Grüninger and M. S. Fox, "Methodology for the Design and Evaluation of Ontologies," 1995.
- [5] D. Vrandečić and A. Gangemi, "Unit tests for ontologies," in *Proceedings of the 2006 International Conference on On the Move to Meaningful Internet Systems*, 2006, pp. 1012–1020.
- [6] S. Peroni, "A simplified agile methodology for ontology development," in *Proceedings of the OWL: Experiences and Directions Workshop and OWL reasoner evaluation workshop*, 2017, vol. 10161, p. 55.
- [7] Y. Ren, A. Parvizi, C. Mellish, J. Z. Pan, K. Van Deemter, and R. Stevens, "Towards competency question-driven ontology authoring," in *Proceedings of the European Semantic Web Conference*, 2014, pp. 752–767.
- [8] C. M. Keet and A. Lawrynowicz, "Test-Driven Development of ontologies," in *Proceedings of the International Semantic Web Conference*, 2016, pp. 642–657.
- [9] S. García-Ramos, A. Otero, and M. Fernández-López, "OntologyTest: A tool to evaluate ontologies through tests defined by the user," in *Proceedings of the International Work-Conference on Artificial Neural Networks 2009*, 2009, pp. 91–98.
- [10] E. Blomqvist, A. S. Sepour, and V. Presutti, "Ontology testing-methodology and tool," in *Proceedings of the Knowledge Engineering and Knowledge Management*, 2012, pp. 216–226.
- [11] A. Lawrynowicz and C. M. Keet, "The TDDonto Tool for Test-Driven Development of DL Knowledge bases," in *Description Logics*, 2016.
- [12] A. Fernández-Izquierdo and R. García-Castro, "Requirements behaviour analysis for ontology testing," in *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2018, pp. 114–130.
- [13] G. Aguado De Cea, A. Gómez-Pérez, E. Montiel-Ponsoda, and M. C. Suárez-Figueroa, "Natural language-based approach for helping in the reuse of ontology design patterns," in *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2008, pp. 32–47.
- [14] M. C. Suárez-Figueroa, S. Brockmans, A. Gangemi, A. Gómez-Pérez, J. Lehmann, H. Lewen, V. Presutti, and M. Sabou, "NeOn D5. 1.1: NeOn Modelling Components," 2007.
- [15] A. Gangemi and V. Presutti, "Ontology design patterns," in *Handbook on ontologies*, 2009, pp. 221–243.
- [16] SmartM2M, "SAREF extension investigation Technical Report (TR 103 411)," 2007.
- [17] "ISO/IEC 30141:2017:Internet of Things (IoT) - Reference Architectures." International Organization for Standardization, Geneva, Switzerland, 2017. [Online]. Available: <https://www.iso.org/standard/65695.html>

²⁵<http://vicinity.iot.linkeddata.es/vicinity/testing.html>

²⁶<http://ontoology.linkeddata.es>

Communication on Requirements Elicitation and Interaction Design through SPIDe

Jean C. S. Rosa^{α,β}, Beatriz B. do Rêgo^α, Filipe A. Garrido^α,
Pedro D. Valente^{β,γ}, Nuno Nunes^{β,δ}, and Ecivaldo Matos^α

^α*Department of Computer Science, Federal University of Bahia – UFBA, Salvador, Brazil*

^β*ITI/LARSyS, M-ITI, Funchal, Portugal*

^γ*University of Madeira – UMA, Funchal, Portugal*

^δ*Tecnico, University of Lisbon – ULisbon, Lisboa, Portugal*

{jean.rosa, beatrizbr, filipe.garrido}@ufba.br, pvalente@uma.pt, nunojnunes@me.com, and ecivaldo@ufba.br

Abstract—Participatory Design has a large number of techniques that can be used for requirements elicitation and interaction design. However, choosing a technique (or set of them) suitable for both processes can be challenging. In this sense, in this paper, we present a semio-participatory methodological process – SPIDe – for requirements elicitation integrated with interaction design, by means of a case study, with the objective of investigate if the results of SPIDe application satisfies the needs and desires of users, and how communication process occurs during SPIDe application. This paper contributes to the use of Semiotics and Participatory Design for requirements engineering and interaction design, to the integration of both areas, and to the SPIDe application.

Index Terms—User participation, human factors, Semiotic Engineering

I. INTRODUCTION

Several human factors influence software conception and development. Some of these factors are research topics in Software Engineering and Human-Computer Interaction (HCI), as for example gender, ethnicity, personality, culture, and social environment [2]. Communication is also one of these factors [2], and one of the current challenges of Requirements Engineering (RE).

Communication is a cultural process and is a study object of Semiotics, which is realized through signs [3]. De Souza [3] argues that the user interface (UI) is a channel of communication in human-computer interaction. But for the UI conception it is necessary to know users’¹ needs, desires, and constraints for a solution, in order words, it is necessary to know the requirements.

A requirement is therefore an attribute which is defined in order to solve a given problem according the users’ needs, desires, and constraints [1]. For requirements to be elicited, there must be communication between users and requirements

engineers². Communication is the key of RE and must be effective [4]. However, for this effectiveness to take place, it is not enough to simply put the engineer in contact with users, but also to use techniques that facilitate communication [4].

From this perspective, Rosa and Matos [13] consider that the users’ effective participation, in the role of design partners, can facilitate communication between designers, engineers, and users, during requirements elicitation and interaction design, which in turn can also facilitate interaction/communication between users and software.

In this paper, we present SPIDe [10]–[13] (a semio-participatory methodological process), and an investigation about *if the results of SPIDe application satisfies the needs and desires of users, and how communication process occurs during SPIDe application*. For this, a case study was carried out and the data was analyzed using open and axial codings of Grounded Theory. This paper contributes to the use of Semiotic and Participatory Design to integrate requirement elicitation and interaction design.

This paper is organized in 5 sections as follows: SPIDe is presented and detailed in the next section. In turn, the research methodology is described in Section 3; the research results and the conclusions are presented in Sections 4 and 5, respectively.

II. SPIDe

To facilitate the communication between the designer and users through the UI, Rosa and Matos [13] suggest that users become interaction co-authors using the semio-participatory process. Thus, Rosa and Matos [13] developed a process for interaction (re)design called SPIDe. SPIDe is (currently) composed of five PD techniques (contextual inquiry, storytelling, brainstorming, braindraw, and think-aloud) and is based on communication-centered design (CCD) [10], as shown in Figure 1. CCD is an interaction design practice based on Semiotic Engineering [13]. Starting from the interaction concept as a communication between the designer and the

DOI reference number: 10.18293/SEKE2019-200

¹We assume as “problem owners”, the customers, subject-interested, (potential) end users, or stakeholders. The term *user* will always indicate this group.

²In this paper, when considering the *engineer*, we take into account subjects that practice the Requirements Engineering and all other professionals who work in the software conception and development, such as system developers and analysts.

users through the UI (as defined by Semiotic Engineering) [3], the interaction design for the CCD is a process of messages construction/manipulation sent from designers to users [13].

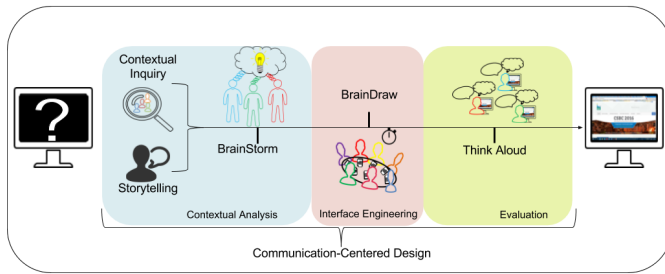


Fig. 1. SPIDe process [10], [13].

1st stage: Contextual Analysis

The first stage is the contextual analysis, which has three PD techniques: *contextual inquiry*, *storytelling*, and *brainstorm*. The contextual analysis stage aims to get to know the users, identify their contexts, characteristics, problems, desires, needs and understand how it is possible to resolve their problem. In addition, it is at this point that the engineer must understand the impact that the solution will produce on users and their environment. This is because the designer is in direct contact with users and their everyday contexts, either by observing the activities carried out by the users through contextual inquiry or listening to their stories through storytelling. At that moment the engineer also knows the socio-cultural context of the users, either through the words told (storytelling) or by immersion in the daily work (contextual inquiry), and together with users, think the problems and possible solutions (brainstorm).

2nd stage: Interface Engineering

The next stage is the interface engineering. This stage has the objective of producing prototypes for the solution envisioned in the previous step. For this, Rosa and Matos suggested the use of *braindraw* [13]. Braindraw is a technique where users draw the software UI in a collaborative way [9], [13]. The users must be divided into groups of at least 2 and at most 8 participants [9]. Color and graphite pencils, pens, erasers are available and each user receives a sheet of paper. Users should draw their ideas of the software UI and at a set time (e.g. 2 minutes) re-pass the paper to those on the left until each paper was drawn twice by each participant.

Through braindraw, users use their signification systems and previous knowledge to draw an UI considered appropriate for use and containing the problem's solution. Developing it in a collaborative way leads to a fusion of ideas, in which all participants are contributors i.e. the drawing contains characteristics of signification systems of their participants/users [9].

3rd stage: Evaluation

Finally, the evaluation stage aims to evaluate the produced prototype. At this time, users use their signification systems to interact with the solution that they have designed, with designer mediation. This stage is composed of the *think-aloud* technique.

For the application of this technique, the designer must design a protocol where the users must follow while interacting with the prototype. During the interaction, the user should speak aloud, as the technique name suggests, about their thoughts, wishes, difficulties, criticisms, suggestions, feelings, and emotions [13]. The interaction should be recorded for further analysis. With think-aloud results, it is possible to identify positive and negative experiences, as well as to identify communication noises that must be repaired before the final version is developed.

A. Researches around SPIDe

The researches presented by Rosa and Matos [13], and Pita et al. [10] specifically deal with interaction design, where communication between designer and user is restricted to the UI (i.e. to HCI). Investigating the communication message composition presented by de Souza [3], Rosa et al. [11], [12] identified that there is a relationship between the message conception and the requirements elicitation process. In this sense, the researchers initiated a project in order to identify if it is possible to elicit requirements through SPIDe.

Initially, a systematic literature review (SLR) was carried out to identify previous research reports that use SPIDe's techniques applied to requirements elicitation [12]. With the SLR, it was identified that only storytelling and brainstorm techniques have already been applied to requirement elicitation; leaving a research gap on the application of other SPIDe techniques, separately or together. In turn, an experimental study was conducted to identify if it is possible to elicit requirements through SPIDe. With the experimental study, the researchers had a positive conclusion [11]. However, this study limits the requirements elicitation, and the researchers did not address how semio-participatory can favor requirements elicitation integrated with interaction design and how the communication happens between designer, engineer, and users. To investigate this gap, in the next section, we present the research methodology.

III. METHODOLOGY

In order to investigate if the results of SPIDe application satisfies the needs and desires of users, and how communication process occurs during SPIDe application, we carried out an exploratory case study [7], [14]. Data was collected using a logbook [6], a semi-structured interview [7], [14], and Technology Acceptance Model Questionnaire (TAM) [5]. The data analysis was done through the open and axial codings of Grounded Theory [15]. The research was based on the construction of software for a music band in which seven subjects, four users and three specialists, participated. This section details the research planning and execution.

A. Research Planning

1) *Research Questions*: To guide the research, the research question is: *is SPIDe effective to make communication feasible for requirements elicitation and interaction design?*

2) *Case Study*: The case study of this research is the software conception for the music band of the Assembly of God church of Ondina, in Salvador/Brazil. The case study was chosen by convenience. The band leader asked the researchers to create a smartphone app to help the band's musicians to perform their activities. The researchers considered that the project was adequate to apply SPIDe and to execute the research, given that in order to construct the demanded solution it was necessary to elicit requirements and also to design the interaction. Complementary, the band members (three subjects) and church pastor³, agreed to participate.

3) *Participants and Roles*: Seven subjects participated in the study⁴, and these were divided into two teams: (a) users and (b) specialists.

All members (three) of the band, Anderson (vocalist and bandleader), Diego (guitarist), and Erick (drummer) and the church pastor (João). The specialists' team was formed by one master and two doctoral students with proven experience in their roles in the software industry. The specialists assumed three roles, namely: SPIDe applicator, requirements engineer, and interaction designer.

4) *Data Collection Procedures and Artifacts*: Three procedures and artifacts were used to collect data. These procedures are widely used in empirical studies of Software Engineering and HCI. The procedures and artifacts are detailed below.

(a) Logbook [6] – The logbook is an artifact where all participants (specialists and users) can take notes, sketches, calculations and describes feelings, ideas, criticisms, suggestions, observations and other information in support of the research.

(b) Semi-structured interview [7], [14] – The semi-structured interview was performed individually, only with the users, after all SPIDe steps were performed, at a moment agreed between users and researchers.

(c) Technology Acceptance Model Questionnaire (TAM) [5] – The TAM aims the identification of the perception about the usefulness and ease of use of SPIDe. The TAM was applied only to the specialists, who answered objective questions on a four-point scale, between totally agreeing and totally disagreeing. Subjective questions were added in order to evaluate negative and positive aspects of SPIDe; aspects of requirements elicitation and interaction design were not contemplated; and the benefit of SPIDe for requirements elicitation and for interaction design in an effective way.

5) *Analysis Procedures*: The data collected through the procedures and artifacts presented previously was analyzed by means of two phases (of the three) of the Grounded Theory [15]. Since the objective of this research is to construct knowledge about how communication occurs through SPIDe, we only used the open and axial codings. Open coding (1st phase) has the objective of creating codes through abstractions

related to the collected data. In turn, in axial coding (2nd phase) the codes created in the open coding are analyzed, and relations between them are established, in order to evidence and discover concepts about the research object [15].

B. Execution

We conducted SPIDe in 4 non-consecutive days of October 2018. In this application, the researchers, in agreement with SPIDe applicator, it was decided not to apply the contextual inquiry, because, in order to apply the technique, the specialists should participate in the band practicing and in church services, which could cause discomfort for both teams, specialists and users.

Initially, it was planned that each stage of the SPIDe would be applied in one day, thus accounting for three days, non-consecutive. However, on the first day of the research, in which the storytelling and brainstorm would be applied, only Anderson and Diego appeared, and in turn, on the second day, João and Erick attended. Faced with the difference of users who attended the first and second days SPIDe applicator decided to reapply storytelling and brainstorm. This impacted on adding one more day to the SPIDe application. In this days the logbook was also distributed. The third day was dedicated to braindraw and the fourth day to thinking-aloud.

Following the thinking-aloud application, the specialists' team answered the TAM and returned their logbook to the researchers. Subsequently, the researchers contacted the users to conduct the interviews. Nevertheless, the researchers were only able to interview users Anderson, Diego, and Erick, because João did not attend the scheduled meetings. The researchers collected the users' logbook, but due to the lack of contact, it was not possible to collect João's. In turn, Erick did not write in his logbook.

IV. RESULTS ANALYSIS

After data collection, this was analyzed using the open and axial codings of the Grounded Theory [15]. We analyze the data collected through logbook, TAM and semi-structured interviews together, to provide a unified view of the users' team and the specialists' team.

Initially, we showed that the communication between the users was effective since they actively participated in the software conception process because they had "[...] *time and freedom to speak*". According to Anderson, "*the fact that we had plenty of time was what helped us [the users' team] to express everything we had to say*". The analysis results indicate that users expressed themselves naturally and "[...] *funny way, helping both design and requirements*".

Users also pointed out that after applying storytelling they recognized their problem with the help of the SPIDe applicator. While they wrote their ideas in brainstorm's post-its, it was noted by the specialists' team, that users could not describe an exact solution. But when they discussed the ideas together with the specialists' team, the solution was clarified and created together, collaboratively. This expresses that through SPIDe it is possible to build mutual knowledge (between users and

³The pastor was invited to participate as he was considered an important stakeholder by the band members and the researchers.

⁴All subjects signed a consent form to participate in the research. The participants also authorized their names publication, except the pastor, who received a fictitious name.

specialists) on the problem and solution like is provided by the PD ideology [8].

In addition to speaking and writing (which corresponded to the data analysis associated with the context analysis), the users' team could also have signs that they consider appropriate to the UI and also use their previous knowledge. This is demonstrated in braindraw sketchings. Through the interview, Diego pointed out that he had put a button on the UI to have a preview of the song they would play. To do this, he was inspired by the music streaming application Spotify®. Another use of previous knowledge was to have the heart to signify that music is a favorite and stars to rank in notes. Both are signs that are already used in other software, are part of the users' signification systems and are built from their relationship with the culture and society they live.

According to the interaction designer, in interface engineering, the users' creativity "[...] has reached significant levels". According to Diego, "*the creation of the application was based on our ideas*". But it was not just the sheet and drawing. With sheet rotation for all participants, as indicated by braindraw [9], it was possible for everyone to draw a part of the UI. Users elected "*the best [UI] design*" democratically, in accordance with what is determined in the PD ideology [8].

After the interface engineering, the specialists turned the drawings into mid-fi prototypes and the users were invited to evaluate it. The specialists have reported that evaluation by think-aloud for some users is not satisfactory. The interaction designer and the SPIDe applicator associated the evaluation with Semiotic Engineering and described signification and communicability problems. In her notes, the SPIDe applicator mentioned that the way in which think-aloud is performed is similar to Communicability Evaluation Method and suggested a change in techniques.

Through the analysis of the collected data, it was still possible to identify that the users consider themselves co-authors of the software. This is because everything has been built collaboratively, between the users and the specialists' team. Thus, according to the research participants, SPIDe reflects what Rosa and Matos [13] thought about the interaction co-design, and goes to a collaborative requirement elicitation. According to the specialists through TAM, the co-authoring provided by SPIDe enables effective communication.

Therefore, SPIDe enables users, their activities, needs, desires, and environments to be (re)cognized (through the use of storytelling and brainstorming) collaboratively; from this, a solution is defined (by means of brainstorm), and also drawn (by braindraw) in a collaborative way. And from the knowledge built in the previous step and with their signification systems, users then interacted with the prototype and evaluated it (through think-aloud).

V. CONCLUSIONS

This paper presented a semio-participatory methodological process SPIDe; and a case study, with analysis via Grounded Theory, to investigate if the results of SPIDe application sat-

isfies the needs and desires of users, and how communication process occurs during SPIDe application.

We concluded that through SPIDe, it is possible for specialists to understand the users desires, needs, and socio-cultural context, given that software conception comes together and initializes from the ideas of its future users. In turn, the problems and solutions that come from the users are clarified as techniques are applied. The specialists' participation is not limited to the observation or techniques application, but these also act in the mediation of communication, to facilitate the mutual construction of knowledge. In this sense, we can conclude that there is evidence in this case study that it is possible to establish effective communication through SPIDe, and achieve satisfactory UI design results, in which users are co-authors. The refinement of SPIDe, as well as the comparison with other processes, are themes for future work.

ACKNOWLEDGMENT

We would like to thank each of the participants of our research, and LARSyS (UID/EEA/50009/2019). This work is supported by the CAPES-DS (Grant #1673896); and CAPES-PDSE (Grant #88881.189073/2018-01).

REFERENCES

- [1] P. Bourque and R. E. Fairley, Guide to the Software Engineering - Body of Knowledge. IEEE Computer Society, 2014.
- [2] A. B. Cunha, A. G. Canen, and M. A. M. Capretz, "Personalities, cultures and software modeling: Questions, scenarios and research directions," in: Proceedings of ICSE Workshop on Cooperative and Human Aspects on Software Engineering, pp. 23-31, 2009.
- [3] C. S. de Souza, The Semiotic Engineering of Human-Computer Interaction. MIT Press, 2005.
- [4] V. V. Das, "Involvement of users in software requirement engineering," in: Proceedings - 10th International Conference on Information Technology, pp. 230-233, 2007.
- [5] F. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," MIS Quarterly 13, 3: 319-340, 1989.
- [6] H. McAlpine, P. Cash, and B. Hicks, "The role of logbooks as mediators of engineering design work," Design Studies 48: 1-29, 2017.
- [7] J. Lazar, J. H. Feng, and H. Hochheiser, Research Methods in Human-Computer Interaction. Cambridge: Morgan Kaufmann, 2017.
- [8] R. Luck, "Dialogue in participatory design," Design Studies 24, 6: pp. 523-535, 2003.
- [9] M. J. Muller, J. H. Haslwanter, and T. Dayton, "Participatory Practices in the Software Lifecycle," in Handbook of Human-Computer Interaction (2nd ed.), M. G. Helander, T. K. Landauer and P. V. Prabhu (eds.), Amsterdam: Elsevier, pp. 256-269, 1997.
- [10] G. L. Pita, D. Zabet, J. C. S. Rosa, and E. Matos, "Adapting the SPIDe to Include Visually Impaired Users in Interaction Design," in Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems - IHC 2017, pp. 1-4, 2017.
- [11] J. C. S. Rosa, E. Matos, F. S. Silva, and G. J. F. Silva, "Experimentando o SPIDe aplicado à Elicitação de Requisitos," in Proceedings of 21st Workshop on Requirements Engineering, pp. 1-14, 2018.
- [12] J. C. S. Rosa, F. S. Silva, G. J. F. Silva, and E. Matos, "Applying SPIDe's Techniques in Requirements Engineering: a systematic review". Systems and Computing Journal 7, 2: pp. 290-303, 2017.
- [13] J. C. S. Rosa and E. Matos, "Semio-Participatory Framework for Interaction Design of Educational Software," in Proceedings of the 15th Brazilian Symposium on Human Factors in Computer Systems - IHC 16, pp. 1-10, 2016.
- [14] F. Shull, J. Singer, and D. I. K. Sjberg, Guide to Advanced Empirical Software Engineering. London: Springer, 2008.
- [15] A. Strauss and J. Corbin, Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. London: SAGE Publications, 1998.

Impact of Agile Practices Adoption on Organizational Learning: a Survey in Brazil

Florindo Silote Neto
LAIS

FUMEC University
Belo Horizonte, Brazil
florindosiloti@gmail.com

Bruno Rafael
de Oliveira Rodrigues

LAIS
FUMEC University
Belo Horizonte, Brazil
bruno.rodrigues@prodemge.gov.br

Renata de Souza França
Fabrício Ziviani

KM INOVA
FUMEC University
Belo Horizonte, Brazil
profrenatafranca@gmail.com
fabricio.ziviani@fumeec.br

Fernando Silva Parreiras
LAIS

FUMEC University
Belo Horizonte, Brazil
fernando.parreiras@fumeec.br

Abstract—Agile software development is a particularly intense knowledge activity in which the success depends greatly on the experience of the professionals involved in the process. Knowledge Management Strategies play an important role in assisting knowledge acquisition and sharing among Agile teams. In this scenario, this paper answer the following research question: What is the impact caused by the use of Agile practices in the process of organizational knowledge acquisition at software development companies? The objective is to analyze strategies for Knowledge Management among teams and evaluate the impact caused by the adoption of Agile practices on the Organizational Learning process. For this, we proposed a model which it was possible evaluate this impact. Thus, a survey was conducted with 455 respondents in order to validate the proposed model. The data collected in this research was processed and analyzed using Structural Equation Modeling. The results corroborates the impact of software development practices on Knowledge Management Strategies and Organizational Learning. Additionally, this study provides mechanisms for software engineering professionals to implement strategies that contribute to the knowledge acquisition and sharing in their teams.

Index Terms—Agile, organizational learning, SEM, agile tailoring

I. INTRODUCTION

Software development teams adopt different approaches for Knowledge Management with the objective of broadening the understanding of individuals, maximizing the productivity of teams and promoting improvements of quality indexes of the projects [1], [2]. In addition, it generates competitive advantage for the company from the application of the available knowledge [3].

Over the last two decades, Agile methods have gained focus in the software engineering research area [4], [5]. Different organizations have changed their processes of software development and adopted Agile practices. However, Agile methods depend on communication and interaction among individuals so that knowledge sharing takes place [6]–[8] and the strategy used is based on customization [9]. In the other words, in practice, Agile methodologies can be combined with traditional approaches, which organizations adopt and customizes the approaches according to their need, using a hybrid software

development approach [10]. In spite of providing a simpler and less bureaucratic process, the Agile methods face difficulties such as the sharing and management of the knowledge the teams had [1], which impact the process of Organizational Learning (OL) at software companies.

So, the present study was guided by the following research question: What is the impact caused by the use of Agile practices in the process of organizational knowledge acquisition at software development companies? The aims of this study are: (1) to investigate which Agile practices are more frequently used by software development teams; (2) to investigate which Knowledge Management Strategies are the most diffused among software development teams that adopt Agile practices; and (3) to propose an empirical model capable of measuring the impact caused by the adoption of Agile practices in the Organizational Learning.

To achieve these goals, this research conducted a survey with 455 professionals from software development companies that utilize Agile methods and practices. The data was collected by using a questionnaire and analyzing it applied to structural equations modeling (SEM). The results demonstrate the possibility of identifying that Agile practices have meaningful influence over the strategies used by the teams in order to share knowledge and affect significantly the Organizational Learning in IT companies. As contribution of this study, we emphasize the importance of Agile practice in the learning process of individuals and organizations. Furthermore, the proposed model represents a breakthrough in literature that lacks empirical studies for Agile methods adoption [11]. We also highlight that the research instrument as well as the parameters used in our research may be reproduced in order to enlarge the comprehension of how Knowledge Management and software engineering correlate, especially when using Agile methods.

The rest of the paper was organized as follows: Section II shows the proposed model. Section III discusses the research method adopted. Section IV presents the results of the study and the Section V presents the threats to validity. The conclusions and future work are presented in Section VI.

II. PROPOSED MODEL

The proposed model in this paper suggests that the adoption of Agile methods affects both Knowledge Management Strategies and the Organizational Learning process. This relationship is justified by the fact that Agile methods are based on learning processes [12]. The use of these methods requires a constant learning stream from teams [13] and Knowledge Management practices are embedded in Agile practices [1], [14]. In this respect, in literature we found the following constructs that compose the model: (a) Agile Adoption, (b) Knowledge Management Strategies and (c) Organizational Learning (OL).

After researching the literature, the construct related to Agile methods adoption was subdivided into two constructs that classify Agile practices into "Project Management Practices" (PMP) and "Software Development Practices" (SDP). We choose this subdivision because agile methods tailoring is a reality in companies that adopt agile methods [15] and the utilization of agile methods as constructs maybe not be suitable for our objectives. This subdivision is based on the justification that methods such as Scrum are more focused on management practices while XP provides more development practices [16], [17]. Moreover, this division provides a method to verify how each proposed Agile practice group impacts the Organizational Learning.

Concerning the "Knowledge Management Strategies" (KM Strategies) construct, the variables used are the strategies that show how organizations promote knowledge sharing [18], [19] and therefore influence the Organizational Learning process [20], [21].

The last construct of this model is the "Organizational Learning" and it has variables of the constructs considering the levels that learning occurs in the organizational environment. [22]. The Figure 1 shows the variables that compose the constructs proposed in this model.

A. Research hypotheses

From the proposed model to conduct this study, 5 hypotheses were drawn up in order to respond to the proposed research question. The proposed hypothetical model:

- **H1:** The adoption of Agile practices for project management has direct influence over the Knowledge Management Strategies used by the teams.
- **H2:** The adoption of Agile practices for project management has direct influence over the process of Organizational Learning.
- **H3:** The adoption of Agile practices for software development has direct influence over the Knowledge Management Strategies used by the teams.
- **H4:** The adoption of Agile practices for software development has direct influence over the process of Organizational Learning.
- **H5:** The adoption of Knowledge Management Strategies influence directly the Organizational Learning process.

The proposed hypothesis H1 and H3 claim that Agile practices adoption by software development teams significantly affect the KM strategies they use. Even though the organizations to where these teams perform have no set Knowledge Management processes, the Agile methods are based on learning processes [12] and the use of these practices contributes to the production and knowledge sharing among team members, since the practices of Knowledge Management are incorporated into Agile practices [1], [14]. In addition to that, software development activities require constant learning and sharing of information as well as cooperation among individuals is crucial for the success of software projects [2].

As for H2 and H4, the model proposed in this study presents a direct influence on Organizational Learning processes when using Agile practices. OL is considered a change that occurs in organizations due to acquired knowledge and experience [22]. This change is identified from the moment individuals in the organization gain new knowledge, new products and services are proposed and also work routines are improved, meaning an alteration in the behaviour of the company [3], [22]. The OL process starts by the production and sharing of knowledge which are activities related to the individual [23]. Once shared, this knowledge produces a common understanding that spreads among the work group [21] and this allows the production and modification of products, services and company routines [3], [22].

In this respect, the adoption of Agile practices encourages the knowledge sharing among individuals [24], [25], meaning experience and knowledge being acquired and shared. Moreover, the adoption of Agile practices and methods require a culture prone to cooperation and knowledge sharing from the organization [4], [24], [25], since a culture led by knowledge sharing is an essential prerequisite for OL to happen [1].

At last, the H5 hypothesis claims that Knowledge Management Strategies directly impact the Organizational Learning process. KM is a process in which the objective is to protect the knowledge resources of an organization [23] and enhance the productivity by means of strategies to knowledge acquisition and sharing [20]. In addition, effective strategies for Knowledge Management provide mechanisms for the production of new knowledge, so that the existing knowledge may be shared among individuals of the organization and the available knowledge turns into a competitive advantage [1], [3]. As a result, knowledge is created by means of Organizational Learning processes managed by Knowledge Management Strategies [21]–[23].

III. METHOD

In order to evaluate the proposed model, a survey questionnaire was performed. A survey questionnaire is suitable for a standardized data collection and allows the researcher to gather relevant information in order to get answers for the research hypotheses [26].

The proposed questionnaire was composed of 38 items subdivided in 4 parts: (1) respondent characterization, (2) use of Knowledge Management Strategies, (3)

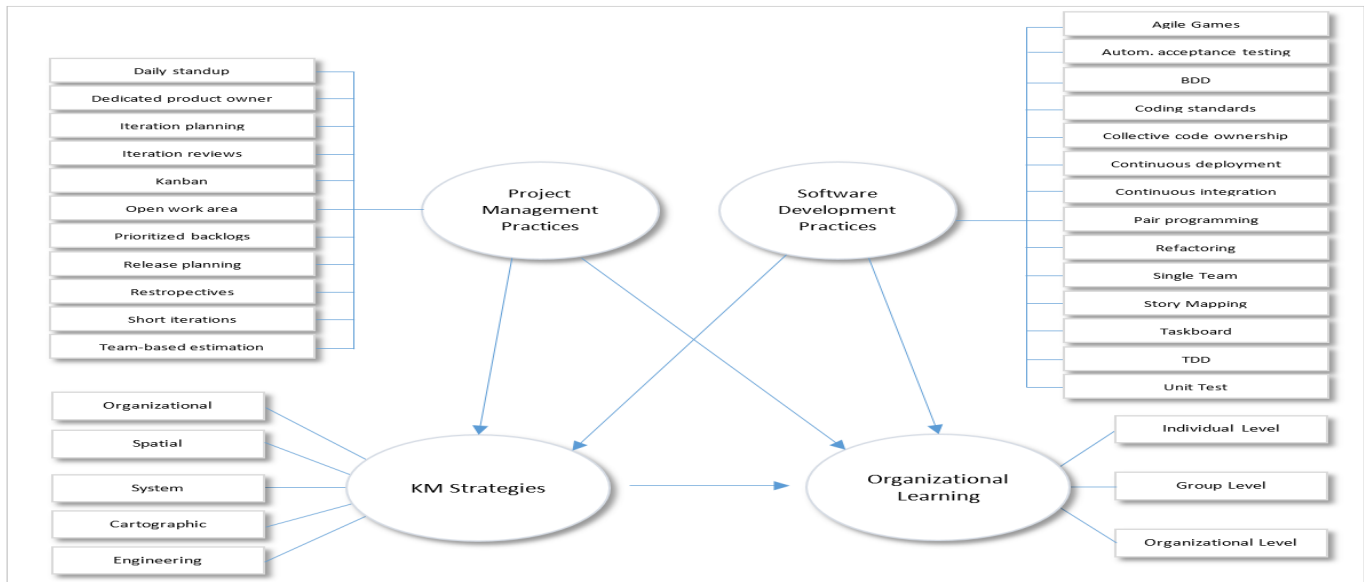


Fig. 1. Research model with constructs and variables.

Organizational Learning and (4) use of Agile practices. Hence, five queries were used to characterize the sample, which include the following item: the individual's job position at the company, schooling, experience with agile methods, level of agile methods knowledge and company stature. The questionnaire was composed by items related to Knowledge Management Strategies, Organizational Learning Agile Practices and it can be seen by link: <https://www.dropbox.com/s/htr6z8ibx1a5pux/sbsi-2019.pdf?dl=0>

IV. RESULTS

This section presents the analyzed results obtained from the data collected utilized for this research.

A. Descriptive analysis of the sample

The survey questionnaire was answered by 455 valid respondents. Among them 52.75% act on the development team as programmers, testers, designers. 30.99% are professionals in management and leadership position such as Scrum Masters and project managers. As for the education, 40.88% of respondents hold graduate degree and 44.62% postgraduate degree or an MBA. 5.05% of individuals mentioned are students in the process of graduating. 11.2% have knowledge on Agile methods, adopt practices on their daily routine but never effectively used these methods in projects. However, most participants are currently embracing Agile methods on projects for their companies, meaning that 20.2% have worked with Agile methods for less than a year and 30.33% have around 1 to 3 years of experience in Agile methods and practices. Emphasizing that 44.62% of respondents consider themselves professionals that have intermediate knowledge on the subject and 26.59% claim to have advanced knowledge on Agile methods. Another highlight is on the fact that 59.12% of individuals work for big companies.

B. Structural Model

To verify the quality of adjustments, the R^2 was used to represent in a scale from 0% to 100% how much the independent constructs explain the dependent ones. Therefore, the values below 25% represent a weak explanatory capacity, the ones between 25% and 50% indicate a moderate explanatory capacity and the values above 50% highlight a substantial explanatory capacity [27].

The Gof value [28] was also used to obtain a geometric average of AVE in all constructs and R^2 from the model. Such measure also ranges from 0% to 100%. It is worth emphasizing that when the PLS approach is used, Gof has no capacity to differentiate the valid models from the invalid ones and must not be applied on models with formative constructs [29]. In this case, Gof allows only a roundup of AVEs and R^2 of the model in one statistics, which could be useful for future adherence comparisons of different samples of the model.

Table I presents the endogenous, which are constructs influenced by other constructs and also introduces the exogenous which are constructs capable of influencing the endogenous ones. Table I shows the values found for the structural model, meaning β value, the standard error for β (S.E.(β)), the Confidence Interval (CI), p-value and R^2 . Highlighting that the proposed model has a Gof of 37.85%. In addition to that, the Confidence Interval was aligned with the results found by p-value, which points out the validity of these results.

TABLE I
STRUCTURAL MODEL EVALUATION

Endogenous	Exogenous	β	S.E.(β)	C.I. - 95%	p-value	R^2
Knowledge Management Strategies	Project Management Practices	0.11	0.06	[0.00 ; 0.24]	0.058	26.5%
	Software Development Practices	0.43	0.06	[0.32 ; 0.55]	0.000	
Organizational Learning	Project Management Practices	0.01	0.06	[-0.11 ; 0.13]	0.859	31.3%
	Software Development Practices	0.16	0.06	[0.04 ; 0.28]	0.007	
	Knowledge Management Strategies	0.45	0.05	[0.35 ; 0.55]	0.000	

Through the analysis of the construct "Knowledge Man-

agement Strategies”, the results indicated a soft (p -value = 0.058) and positive ($\beta=0.11$; [0.00; 0.24]) influence of the construct ”Project Management Practices” (PMP) over the construct ”Knowledge Management Strategies”. In this case, it means that the higher the usage of Agile practices for project management, the higher the usage of strategies for Knowledge Management will be. Moreover, there was a meaningful (p -value = 0.000) and positive ($\beta=0.43$; [0.32; 0.55]) influence of the construct ”Software Development Practices” (SDP) over the construct ”Knowledge Management Strategies”. This influence is directly proportional, which means the higher Agile practices to software engineering or software development ones are, the higher the Knowledge Management Strategies will be. Thus, the constructs ”Project Management Practices” and ”Software Development Practices” were capable of explaining 26.50% of the variability of the construct ”Knowledge Management Strategies”, which means that there is a moderate explanatory capacity. These data are presented on Figure 2 describing the structural model of this analysis.

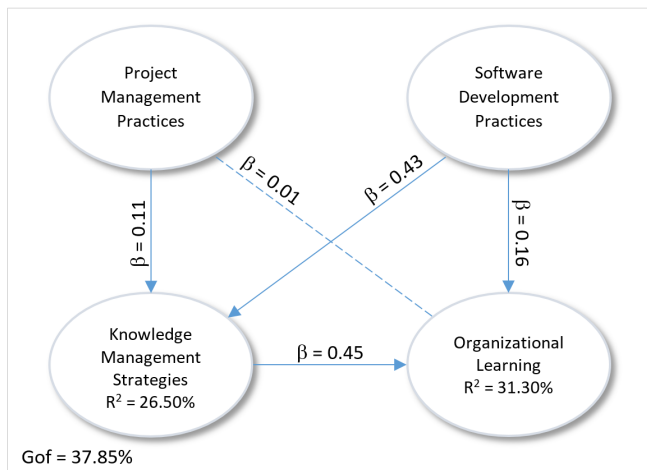


Fig. 2. Structural Model Presentation.

Regarding the construct ”Organizational Learning”, there was no meaningful influence of the ”Project Management Practices” ($\beta=0.01$; [-0.11; 0.13]). On the contrary, the results demonstrated meaningful and positive influence (p -value = 0.007) of the construct ”Software Development Practices” ($\beta=0.16$; [0.04; 0.28]) over the ”Organizational Learning”. Hence, having high SDP affects directly the Organizational Learning which tends to be high as well. A meaningful and positive influence (p -value = 0.000) of construct ”Knowledge Management Strategies” ($\beta=0.45$ [0.35; 0.55]) was also identified over the construct ”Organizational Learning”. Consequently, the growth of ”Knowledge Management Strategies” imply the growth of ”Organizational Learning”. The PMP, SDP and KM Strategies were able to explain 31.30% of the variability in ”Organizational Learning”, which indicates a moderate explanatory capacity. These data are presented in Figure 2.

C. Analysis of hypothesis

In this section, the proposed hypothesis is discussed and the results are confronted with the literature presented in the introduction of this work.

As presented on Section IV-B, the hypotheses H1 and H3 referring to the influence of the Agile practices adoption over Knowledge Management Strategies, were confirmed. The marginally significant and positive influence between constructs ”Project Management Practices” and ”KM Strategies” confirm the hypothesis 1. Furthermore, the meaningful and positive result found between constructs ”Software Development Practices” and ”KM Strategies” confirm hypothesis 3.

Thus, it is possible to state that Knowledge Management Strategies used by teams that adopt agile methods have direct influence due to the set of Agile practices employed by these teams. It is worth mentioning that the results of the study showed that Agile practices such as ”practices for software development” [16], [30] present a higher influence over Knowledge Management Strategies. The practices in this group also present further alignment with social aspects concerning KM and provide mechanisms aimed at individual learning [5], [16].

By assessing the results from the H1 and H3 hypotheses, it is possible to relate them with the results obtained in others researches [14], [31]. The hypotheses H2 and H4 refer to the influence of Agile practices in the Organizational Learning process. The results obtained through the questionnaire confirmed only the H4 hypothesis, which points to a meaningful influence of the construct ”SDP” over the construct ”Organizational Learning”. In this respect, the use of Agile practices for software development has direct influence over the process of Organizational Learning although such influence was not confirmed for Agile practices for project management (H2). The confirmation of the H4 hypothesis may be for the reason that the Software Development Practice (SDP) encourage the knowledge sharing among individuals [24], [25] and provide a set of practices aligned with the social aspects of learning [5]. Additionally, Agile practices and methods require a change in the culture of the organization, that is, it must be guided by learning and constant update [16].

The non-confirmation of the H2 hypothesis diverges from the results reported by authors that consider practices for project management, such as daily meetings and retrospectives, as efficient mechanisms for the process of Organizational Learning [32]. Schwaber and Beedle [31] state that knowledge sharing takes place by four Agile practices of project management (sprint planning, daily meeting, sprint reviews and retrospectives). However, Hoda, Babb and Nørbjerg [13] emphasize that in an environment under pressure for results and deliveries compromise learning due to the lack of ceremonies and Agile practices related to learning.

The confirmation of these hypotheses (H1, H3 and H4) converges with the discussion regarding the nature of Agile methods and practices, which means that agile methods are based on learning processes [12] and Knowledge Management

practices are incorporated into agile practices [1]. Furthermore, software development activities demand that teams constantly produce new knowledge [13] and share it so that success is achieved in software projects [2], contributing for product construction and improvement of the processes kept by organizations.

Finally, the last hypothesis proposed by the hypothetical model (H5) establishes that the use of strategies for Knowledge Management influence directly the process of Organizational Learning. The results from this study lead to a meaningful influence of construct “KM Strategies” over the construct “Organizational Learning”, which confirms the hypothesis 5. The application of appropriated Knowledge Management Strategies provide mechanisms for Organizational Learning to take place, meaning that new knowledge is produced and shared among individuals [1], [3]. This process produces a suitable environment for the development of new products and services, and besides that provides means for the improvement of the routines of the organization [21], [22].

D. Discussion

From the results, it was possible to identify the goals of this research. First, (1) to investigate which Agile practices are more frequently used by software development teams, it was verified the frequency in which professionals use practices described as “Agile practices for project management”. As for practices described as “agile practices for software development”, the results indicated that they are not often used. Only the practices “Unit Testing”, “Taskboard”, “Single Team”, “Continuous Integration”, “Collective Code Ownership” and “Coding Standards” were described as usual practices in the daily work of teams. Even though this result demonstrates that most Agile practices of software development are rarely used by the teams, the results from the present study converge with other researches done by companies from the IT area [30]. This outcome helps us reach the first goal of this paper, which consists of identifying the frequency on which Agile practices are used by teams.

Second, (2) to investigate which Knowledge Management Strategies are the most diffused among software development teams that adopt. The data collected in this research demonstrated the Knowledge Management Strategies described as System, Engineering, Organizational and Spatial are present in companies that adopt these methods. Only the use of strategies from Cartographic school [20] got impartial values for the respondents, which demonstrates that the companies do not map the competencies of employees. Even though the participants of this research identified and agreed to most KM strategies, we also identified strategies established by the Behavioral school (Spatial and Organizational) with bigger indexes of concordance than the ones from the Technocratic schools (System and Engineering). These results are justified by the characteristic of the sample obtained through the questionnaire, which was composed by professionals who work in teams that have adopted Agile practices and methods. These results are in conformity with the results of other authors who

showed the strategies predicted by Behavioral schools as the most used ones in Agile environments [18], [19], [33], while Technocratic strategies are more present in organizations that use the traditional methods [19].

The last objective, (3) to propose an empirical model capable of measuring the impact caused by the adoption of Agile practices in the Organizational Learning. The fulfillment of this objective allows to answer the proposed research question. From the analysis of the collected data, it was concluded that these practices for project management do not influence significantly the Organizational Learning process. However, the results show that Agile practices for software development have positive and meaningful influence on Organizational Learning. It was also possible to identify that the use of KM strategies have meaningful and positive influence on Organizational Learning. Therefore, agile practices and Knowledge Management Strategies enable to explain the 31.3% of variability in Organizational Learning in software companies. In addition, highlighting that Agile practices (for project management and software development) have meaningful influence on strategies used by Agile teams to Knowledge Management and sharing. The use of Agile practices makes it possible to explain 26.5% of the variability in Knowledge Management Strategies used by the teams.

V. THREATS TO VALIDITY

In this paper, some threats to validity have been identified and some measures have been adopted to mitigate them. The first, the data collection instrument used for this study. There was the possibility that the participants of the research could find difficulties to understand or fill in the questionnaire. In order to prevent this from happening, a previous test was made with the individuals in the same parameters from the original research. From this test, it was possible to measure the average time spent on answering the questions as well as identifying and discussing points for improvement. The items indicated by the individuals taking part in the test were adjusted in the data collection instrument.

The second, the possibility of professionals who were not aware about Agile methods and practices to answer the questionnaire and compromise the data collected. In order to mitigate this, five questions were included in the data collection instrument to characterize the sample. Thus, it was possible to remove from the research database the questionnaires in which participants affirmed having no knowledge concerning Agile methods and practices.

A hypothetical model was proposed seeking to validate it from an empirical research that allows for the generalization of the results obtained. For this reason, the regional character of the data collection was an issue to the research, since the obtained results could express the characteristics of one region of the country instead of the whole software production sector. To prevent this, other than the snowball technique, data collection has been made in three different events in two capitals. Even though data about the work place has not been asked from participants, the events chosen for data collection

have national expression and receive audience from all over the country, especially the Agile Trends event that took place in São Paulo at the occasion of the data collection.

VI. CONCLUSION

In this study, we proposed a hypothetical model composed by four constructs and conducted a survey with 455 respondents to enable the validation of the the proposed model. The data collected was analyzed from the technique of Structural Equations Modeling (SEM). Thus, with the results of this work it's possible verify that Agile practices have positive and meaningful influence on OL, KM strategies have meaningful and positive influence on Organizational Learning and the Agile practices have meaningful influence on strategies used by Agile teams to Knowledge Management and sharing.

As future work, we suggest the validation of the proposed model, collecting data in companies previously chosen and case studies in these organizations in order to engage in qualitative evaluation when interviewing experienced professionals in Agile methods and practices. Thus, the results obtained in this work may be confronted with the reports of the professionals interviewed. Nevertheless, we can not generate the findings of this study, because the present research was conducted in a specific country, Brazil. Therefore, replication of this survey in others countries is also recommended.

REFERENCES

- [1] R. Kavitha and M. Irfan Ahmed, "A Knowledge Management Framework for Agile Software Development Teams," in *Process Automation, Control and Computing (PACC)*, 2011 International Conference on, Jul. 2011, pp. 1–5.
- [2] N. Porrawatpreyakorn, W. Chutimaskul, G. Quirchmayr, and M. Sodanil, "A Knowledge Transfer Framework for Supporting the Transition to Agile Development of Web Application in the Thai Telecommunications Industry," in *Proceedings of International Conference on Information Integration and Web-based Applications & Services*. New York, NY, USA: ACM, 2013, pp. 140:140–140:148.
- [3] P. Zappa and G. Robins, "Organizational learning across multi-level networks," *Social Networks*, vol. 44, pp. 295–306, Jan. 2016.
- [4] R. T. Nishijima and J. G. Dos Santos, *the Challenge of Implementing Scrum Agile Methodology in a Traditional Development Environment*. Council for Innovative Research, 2013.
- [5] F. S. Santos and H. P. Moura, "Analyzing the Intertwining of Social and Technical Aspects in Agile Methods," in *Social Informatics (SocialInformatics)*, 2012 International Conference on, Dec. 2012, pp. 320–327.
- [6] S. Ryan and R. V. O'Connor, "Acquiring and sharing tacit knowledge in software development teams: An empirical study," *Information and Software Technology*, vol. 55, no. 9, pp. 1614–1624, Sep. 2013.
- [7] H. Holz and J. Schafer, "Collaborative, task-specific information delivery for agile processes," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*. IEEE, 2003, pp. 320–325.
- [8] C. Loftus and M. Ratcliffe, "Extreme Programming Promotes Extreme Learning?" in *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: ACM, 2005, pp. 311–315.
- [9] M. T. Hansen, N. Nohria, and T. Tierney, "What's your strategy for managing knowledge?" *Harvard Business Review*, vol. 77, no. 2, pp. 106–116, 187, Apr. 1999.
- [10] M. Kuhrmann, P. Diebold, J. Münch, P. Tell, V. Garousi, M. Felderer, K. Trektore, F. McCaffery, O. Linssen, E. Hanser, and C. R. Prause, "Hybrid software and system development in practice: Waterfall, scrum, and beyond," in *Proceedings of the 2017 International Conference on Software and System Process*, ser. ICSSP 2017. New York, NY, USA: ACM, 2017, pp. 30–39. [Online]. Available: <http://doi.acm.org/10.1145/3084100.3084104>
- [11] K. Kuusinen, P. Gregory, H. Sharp, L. Barroca, K. Taylor, and L. Wood, "Knowledge sharing in a large agile organisation: A survey study," in *Agile Processes in Software Engineering and Extreme Programming*, H. Baumeister, H. Lichter, and M. Riebisch, Eds. Cham: Springer International Publishing, 2017, pp. 135–150.
- [12] S. Kizaki, Y. Tahara, and A. Ohsuga, "Software Development PBL Focusing on Communication Using Scrum," in *Advanced Applied Informatics (IIAIAI)*, 2014 IIAI 3rd International Conference on, Aug. 2014, pp. 662–669.
- [13] R. Hoda, J. Babb, and J. Nørhjerg, "Toward Learning Teams," *Software, IEEE*, vol. 30, no. 4, pp. 95–98, Aug. 2013.
- [14] A. Singh, K. Singh, and N. Sharma, "Agile knowledge management: a survey of Indian perceptions," *Innovations in Systems and Software Engineering*, vol. 10, no. 4, pp. 297–315, 2014.
- [15] A. S. Campanelli and F. S. Parreiras, "Agile methods tailoring – A systematic literature review," *Journal of Systems and Software*, vol. 110, pp. 85–100, Dec. 2015.
- [16] S. Lee and H. Yong, "Agile Software Development Framework in a Small Project Environment," *Journal of Information Processing Systems*, vol. 9, no. 1, pp. 69–88, Mar. 2013.
- [17] A. R. Y. Cabral, M. B. Ribeiro, and R. P. Noll, "Knowledge management in agile software projects: A systematic review," *Journal of Information & Knowledge Management*, vol. 13, no. 1, 2014.
- [18] M. A. Razzak and D. Smite, "Knowledge Management in Globally Distributed Agile Projects – Lesson Learned," in *Global Software Engineering (ICGSE)*, 2015 IEEE 10th International Conference on, Jul. 2015, pp. 81–89.
- [19] T. Dingsøyr, F. Bjørnson, and F. Shull, "What Do We Know about Knowledge Management? Practical Implications for Software Engineering," *IEEE Software*, vol. 26, no. 3, pp. 100–103, May 2009.
- [20] M. Earl, "Knowledge management strategies: Toward a taxonomy," *Journal of Management Information Systems*, vol. 18, no. 1, pp. 215–233, 2001.
- [21] G. P. Huber, "Organizational learning: The contributing processes and the literatures," *Organizational learning*, pp. 124–162, 1996.
- [22] L. Argote, *Organizational learning creating, retaining and transferring knowledge*. Boston, MA: Springer US : Imprint: Springer, 2013.
- [23] L. Iebra Aizpurúa, P. E. Zegarra Saldaña, and A. Zegarra Saldaña, "Learning for sharing: an empirical analysis of organizational learning and knowledge sharing," *International Entrepreneurship and Management Journal*, vol. 7, no. 4, pp. 509–518, Dec. 2011.
- [24] K. Beck and C. Andres, *Extreme programming explained: embrace change*, 2nd ed. Boston, MA: Addison-Wesley, 2005.
- [25] K. Schwaber and J. Sutherland, "The definitive guide to scrum: The rules of the game," <http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf>, 2013, accessed: 2016-04-21.
- [26] J. F. Hair, R. L. Tatham, R. E. Anderson, and W. Black, *Multivariate data analysis*. Pearson Prentice Hall Upper Saddle River, NJ, 2006, vol. 6.
- [27] J. F. Hair, G. T. M. Hult, C. Ringle, and M. Sarstedt, *A Primer on Partial Least Squares Structural Equation Modeling (PLS-SEM)*, 2nd ed. CA, USA: Sage Publications, 2016.
- [28] M. Tenenhaus, S. Amato, and V. E. Vinzi, "A global goodness-of-fit index for pls structural equation modelling," in *In Proceedings of the XLII SIS scientific meeting*, vol. 1, 2004, pp. 739–742.
- [29] J. Henseler and M. Sarstedt, "Goodness-of-fit indices for partial least squares path modeling," *Computational Statistics*, vol. 28, no. 2, pp. 565–580, 2012.
- [30] VersionOne, "The 11th annual state of agile report," <http://stateofagile.versionone.com>, 2017, accessed: 2017-06-03.
- [31] S. Dorairaj, J. Noble, and P. Malik, "Knowledge Management in Distributed Agile Software Development," in *Agile Conference (AGILE)*, 2012, Aug. 2012, pp. 64–73.
- [32] M. A. Razzak and R. Ahmed, "Knowledge sharing in distributed agile projects: Techniques, strategies and challenges," in *Computer Science and Information Systems (FedCSIS)*, 2014 Federated Conference on, Sep. 2014, pp. 1431–1440.
- [33] F. O. Bjørnson and T. Dingsøyr, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," *Information and Software Technology*, vol. 50, no. 11, pp. 1055–1068, Oct. 2008.

Towards a customizable Student Information System integrating MDD and SPL

Vale, A., Fernandes, S., Magalhães, A. P.

Post-Graduation Program in Systems and Computing (PPGCOMP) Salvador University (UNIFACS)
Salvador, Bahia, Brazil

andersoncunh@gmail.com, sergiomfernandes63@gmail.com, anapatriciamagalhaes@gmail.com

Abstract – Student evaluation criteria is a non-trivial variation point in a Student Information System software product line. It can vary significantly in different institutions, or even in different areas of a specific institution, and also over time. This paper presents a highly flexible model-driven development solution to incorporate and change one or many different evaluation criteria in each specific product of the line, where a domain expert should be able to model each set of student evaluation criteria, and automatically transform it into software code, without involvement of software engineers. For this purpose, we created a domain-specific modeling language DSCHOLAR and a model-to-code transformation *dscholar2Code* to model and automatically implement such a software component. They were both validated using a case study and proof of concept, respectively.

Keywords – Domain-Specific Modeling Language; MDD; Academic Application; Student Evaluation Criteria, Student Information System.

I. INTRODUCTION

Around the world, the evaluation criteria of higher-level education institutions students vary significantly among countries, institutions, different areas of the same institution, and over time.

External factors can impact an institution's evaluation criteria. In some places, the government applies country wide student evaluations to measure the quality of its institutions. The institutions may respond by performing internal simulations of the government evaluation, and the results of these simulations are sometimes factored, in some way, into the grade students get for each individual subject they are taking. Each institution will use some specific strategy to do this, in ways that stimulate students to improve their performance in government evaluations.

Internal factors can also impact the institution evaluation criteria. In one university, for instance, some courses (but not all of them) have an entrepreneurship project whose evaluation must be factored into the students' regular classes grades – in different ways depending on the semester the student is in.

Another factor is the dissemination of distance learning under graduate courses, where, typically, the flexibility of evaluation criteria is more pronounced.

Also, some institutions have the same rules for all its students, while others have rules for each specific course or even rules defined by each teacher individually.

The context for this work is the design and development of a Student Information System (SIS) for the management of academic and support functions of higher-level educational institutions. It is designed to be a software product family (SPL) [1] for deployment in many institutions, with many common functionalities, but also significant non trivial variability points, such as the evaluation criteria. Therefore, each product of the family can be adapted for the specific needs of the educational institution and can also easily evolve.

This paper focuses specifically on the solution created for one specific variability point of the product line: student evaluation criteria in higher-level educational institutions.

Many approaches deal with variability in product families. We used a model-driven development (MDD) approach [2] to model the evaluation criteria component of our SPL, according to the institution's needs, and to generate its code automatically. In order to support it, a small domain-specific modeling language (DSML) was developed specifically for the domain of evaluation criteria of educational institutions, and a model-to-code transformation was developed to automatically translate the models produced in this DSML into C# language software code.

The MDD / DSML software tool selected for this project was Microsoft DSL Tools [7], a set of plugins hosted by Microsoft Visual Studio that comprises a wizard and a graphical environment for DSML creation and editing; a DSML validation engine; and a transformation generator that creates transformations that translate DSML models into high-level programming language code.

The rest of the paper is organized as follows: section II briefly introduces SPL and MDD approach as well as the technology used in the project; section III discusses some related works; section IV describe the DSML and its validation respectively; section V details the transformation; section VI presents the solution validation; and section VII presents the conclusions and future works.

DOI reference number: 10.18293/SEKE2019-089

II. BACKGROUND

Among the approaches proposed to increase the productivity of development processes and the quality of software products, SPL and MDD stand out particularly in a context of an information system that needs non-trivial and extensive customization for each specific customer as it evolves.

A. *Software Product Lines and Model-Driven Development*

An SPL is a set of software products with similar features that share a common infrastructure and the parameterization of differences among the products [3]. The features of an SPL are usually classified as mandatory or optional and can be used to specify variabilities and commonalities among software products. The development of software using SPL is based on a set of core assets, defined according to the commonalities and variabilities of a specific domain, used to derive new products of the line.

MDD is a software development approach based on higher abstraction level models, and (semi) automatic transformation of these models into other lower abstraction level models, and frequently, in the end, into source code [4]. These models are typically expressed as UML diagrams or by using DSMLs. Therefore, the use of MDD requires the definition/adoption of a modeling language as well as transformations to convert models, instances of the modeling language, into other models and code.

Those who propose DSML-based variants of MDD typically argue that software representation through DSMLs is more expressive and more straightforward than using general-purpose modeling languages, such as UML, and can be used by a broader universe of professionals, particularly domain experts, not just experts in UML modeling.

B. *SIS product family*

A SIS “consists of several basic functional modules to support features such as system setup (e.g. managing users, roles, countries, buildings, rooms, faculties, and departments), academic setup (e.g. managing courses, course sections, and study plans), admissions, student record management (e.g. managing personal information, scholarships, schedules, grades, transcripts, major transfers, etc.), registration (i.e. adding and dropping courses), final exam scheduling, grade processing (i.e. entering grades, computing Grade Point Averages (GPA), and viewing transcripts), graduation, and reporting” [5].

A SIS can be best understood as a product family because different institutions will have many similar software needs but also many typically non-trivial specificities.

III. RELATED WORKS

MDD has been used to develop the variable parts of a product in an SPL for more than ten years, usually in telecommunications, banking, embedded systems and automotive domains [8].

Many authors propose development approaches based on SPL integrated with MDD through which models are generated

using DSMLs. Among them, [9] propose the use of DSMLs to configure instances of product families. [10] argue that building a language and framework for a narrow domain makes more economic sense in the context of a software product family. Multiple DSMLs are required to describe a business application. [11] propose the use of DSMLs to deal with both product family variability and that of a single product over time. [1] claim that DSMLs are useful to define the specific configuration of a particular member of a family of systems. They also consider that, in general, it is necessary to define several different DSMLs for a complete application. [12] argues that libraries or frameworks are alternatives to building a DSML and that a DSML should only be developed when there is sufficient domain knowledge and conventional programming techniques fail to provide adequate abstraction mechanisms.

Regarding specific case studies that integrate SPL and MDD, [13] propose FArM (Feature-Architecture Mapping) method, which is based on a series of transformations that generate architectural components that encapsulate the business logic of each transformed feature, and the interfaces directly reflect the interactions of the feature. [14] presents a case study in the automotive domain, comprising guidelines for the development of architectural transformations on a model that represents different points of view of a system. [15] performed a case study in the health domain to illustrate the transformation process for product generation of an SPL. They used DSMLs to model the architecture and the application. Then they modeled points of variability according to the needs of some users who used the application and automatically transformed those models into products with the requirements requested by the users.

In the same way as the works mentioned above, we integrate SPL and MDD. However, we apply this strategy to support the development of systems in the education domain, where, to the best of our knowledge, it has not been used before.

IV. DSCHOLAR DSML

This section presents the DSML defined to support the development components for evaluation criteria using MDD.

DSCHOLAR is a domain-specific modeling language defined as part of our MDD solution to represent the domain of student evaluation criteria used in the SIS product line. To create the DSCHOLAR metamodel, we used the process proposed by [16]. It is bottom-up, iterative and incremental, where we start with examples, define one version of the metamodel based on those examples, and iterate through other examples, and so on.

DSCHOLAR’s abstract syntax is presented in figure 1, where a model that represents a specific evaluation criterion comprises one *Entity* and many *Evaluations*

An *Entity* represents the area to which the same evaluation criteria applies, such as a university or department of a university – or even a single class of an academic subject. An *Evaluation* represents the specification of an evaluation that is applied by the *Entity*, in each academic period. Each *Evaluation* has four attributes: *name*, *weight* (the relative weight of one evaluation relative to the others), *description* and *sequence* (the sequence in time of each evaluation). The relationship between an *Entity* and

their corresponding *Evaluation* is defined through the association *EntityReferencesEvaluation*.

In DSCHOLAR, *Evaluation* is a general concept, specialized by five other concepts: *Mandatory Evaluation*, which must be applied; *Optional Evaluation*, which is part of the evaluation criteria but that may be applied at the teacher's discretion; *Various Evaluations*, that applies when teachers are free to define a number of evaluations not predefined by the evaluation criteria; *Extra Evaluation*, which is a special evaluation whose grade is to be added to that of another evaluation grade; and *FinalEvaluation*, which is applied only when the student does not reach the Entity average.

There is a composition relationship between an *Evaluation* and itself. This means that the grade of a student in a particular evaluation may be a composition of grades of sub-evaluations, each with its own specific weight.

Some OCL constraints (not showed here due to a lack of space) were specified in order to guarantee model integrity, e.g. to ensure that the sum of the weights of sub evaluations corresponds to the weight of the super evaluation.

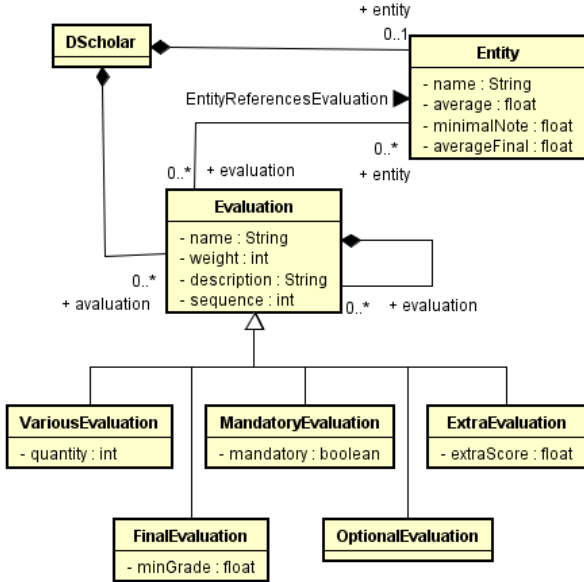


Figure 1 - Abstract Syntax of DSCHOLAR

Figure 2 shows an example of a model defined for a specific university. As can be seen, there is one *Entity*, named *Private Institution* and three *Evaluations*, named *Evaluation1*, *Evaluation2* and *Evaluation3*, with their respective relative weights (30, 40, 30). In this example, all the courses at the *Private Institution* adopt the same evaluation criteria. The round-cornered rectangle of the first two evaluations as well as the Boolean values (true) at the bottom of them is the concrete syntax used to specify that all of them are mandatory evaluations. The third evaluation is an optional one, which is depicted by a rectangle in a different color.

Regarding the number of *Optional Evaluations* in a model, there are two different modeling options. If the quantity of optional evaluations is already defined for an *Entity*, each one of these evaluations is represented by an instance of an *Optional*

Evaluation modeling element in the respective model. Or, each teacher can define their quantity so that the model will have only one instance of *VariousEvaluations* and an attribute *quantity* is used to define the upper boundary of this quantity.

In Figure 2 the grade of *Evaluation 2* is a composition of *Test Evaluation*, *Arhte Evaluation* and *AIC Evaluation*, each with their respective weight. The *Arhte Evaluation* is an extra one (with a different color and small icon), which means that its grade will be added to the grade calculated by the weighted mean of the other sub-evaluations of *Evaluation 2*. For example: if the weighted mean of *Test Evaluation* and *AIC Evaluation* is 8 for a student, and the grade of *Arhte Evaluation* is 1 for the same student, the student's final grade in *Evaluation 2* will be 9.

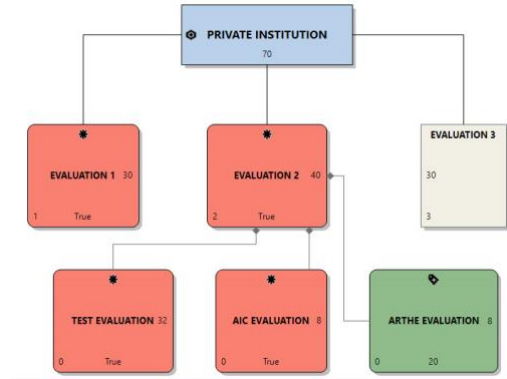


Figure 2 - Evaluation criteria model for a private University

V. CODE GENERATOR

This section presents the transformation, named *dscholar2Code*, developed to support code generation of the component Evaluation Criteria of the SIS. The transformation receives as input a DSCHOLAR model specifying the evaluation criteria of a specific institution or one of its units and generates as output the correspondent code in C# language.

A five stages process was defined to specify the transformation *dscholar2Code*. The first stage, named *Product Design*, concerns the architecture and design definition. The MVC (Model-View-Controller) architectural pattern [18] was used because it is well suited for a web information system. For the component *Evaluation Criteria*, the Model layer was specified using the DSCHOLAR metamodel as input.

The second stage, *Implementation Strategy Definition*, deals with the identification, in the class structure modeled by the previous step, of the elements of each class which are variable or static. The variable elements must be dynamically generated, and the static ones are generated manually. Based on this, for each class, the code snippets which should be generated automatically, and which snippets should be static are determined.

For the component Evaluation Criteria, the templates defined for the View layer are also dynamically customized using the DSCHOLAR model as input. The Control layer is manually generated as it does not vary according to the evaluation criteria. The Model layer comprises dynamically and statically generated code. The code to be statically generated was the structural part of the class *Entity* and the declaration of its attributes, such as

entityName, *meanGrade*, *lowestGrade* and *finalMeanGrade*. The part to be dynamically generated was the methods *loadEntity()* and *generateEvaluationsList()* because they contain information that varies according to the student evaluation criteria of the input model.

At stage 3, *Transformation Rules Definition*, transformation rules must be specified. They map the elements of DSCHOLAR and the bits of code that are dynamically processed by the transformation *dscholar2Code*. The transformation code reads a model and manipulates its elements through tags in order to dynamically generate code that is inserted in the respective template source code. Therefore, in order to specify transformation rules, we map each relevant element of DSCHOLAR metamodel into a tag in the transformation code.

Stage four is called *Transformation Implementation*. The language used to implement the transformation code is based on templates, where a predefined template contains the code parts which are static as well as the specific points where the dynamic code must be inserted when generated.

A loop-like programming structure was used, which reads a model (like the one shown in Figure 3) in search of instances of the *Entity* and *Evaluations* elements as well as their attributes in the input model. Figure 4 shows the implementation of the method *generateEvaluationsList()*.

```
Public void loadEntity(){
    <# foreach(Entity ent in this.Dscholar.Entity){ #>
        this.entityName = "<# ent.name #>";
        this.meanGrade = <# ent.meanGrade #>;
        this.lowestGrade = <# ent.lowestGrade #>;
        this.finalMeanGrade = <# ent.finalMeanGrade #>;
    <# } #>
}
```

Figure 3 - Part of the code of the method *loadEntity()*

The method *loadEntity()* assigns a value to each attribute of each object of the *Entity* class existing in the input model. The code in Figure 3 illustrates the search for an instance of type *Entity* in a DSCHOLAR model, and the storing of its attributes *entityName*, *meanGrade*, *lowestGrade* and *finalMeanGrade* in the instance of the C# *Entity* class being generated.

```
public void generateEvaluationList(){
    <# foreach (Evaluation av in this.Dscholar.evaluations){ #>
        <# if (av.Targets.Count == 0){#>
            <# if (av.GetType().GetProperty("mandatory") !=
            null){#>
                this.evaluations.Add(new Evaluation("<# av.name
            #>",<# av.weight #>",<# av.description #>",<# av.sequence
            #>, 1 ));
            <# } } }#>
        }
}
```

Figure 4 - Code of the method *generateEvaluationList()*

The code of the method *generateEvaluationList()* (Figure 4) is dynamically generated based on the list of *evaluations* that are part of each entity in the input model. As a result, it will fulfill a list in the C# code (named *evaluations*) which contains all the corresponding evaluations of the input model. When this generated code is executed it scrolls the list instantiating each one of the evaluations.

Once the transformation is implemented, it is tested

(*Validation Transformation*, stage 5 in our method). This is described in the next session.

VI. PROPOSAL EVALUATION

In order to evaluate our proposal we performed a case study, to evaluate the expressiveness of DSCHOLAR in specifying evaluation criteria in different scenarios, and a proof of concept, to evaluate the correctness of the code generated by the transformation *dscholar2Code*.

The case study consisted of the specification of different evaluation criteria at four universities. To assist the validation, we followed the metamodel design method proposed by [16] and the guidelines for software engineering experimentation presented in [17]. The questions underlying the validation are: Q1. Do the language constructors sufficiently specify all the needs of a university evaluation criteria? Q2. Is it necessary to add new constructors in the DSML to enable the specification of different evaluation criteria scenarios?

Data collection was done using a direct method, through the application of a questionnaire during the execution of the study; and an independent method, through the analysis of the documentation produced by the participants, i.e. a model, written in our DSML. The metric used to evaluate the study was *DSML coverage*, which was measured two indicators: *#UC* (used constructors) and *#MC* (missing concepts). *#UC* measures the number of DSML constructors used in a model, collected from the model produced by the teaching staff; *#MC* measures the number of concepts present in the university academic evaluation criteria that could not be modeled by our DSML, collected in the questionnaire answered by participants. *#UC* is used to identify how many constructors as well as which of them has been validated through the study. *#MC* is used to improve the DSML. The goal is that after some validations, the *#MC* becomes zero, indicating that the metamodel covers the definition of many kinds of evaluation criteria.

As we iteratively validated and modified the DSML during this study, at the end of the fourth validation, we observed that all the concepts defined in our DSML were used in the models produced, i.e. *#UC=100%*. Therefore, we can say that it covers the necessary constructors to instantiate the models (related to Q1). Moreover, the missing concepts identified during the process were included and are now part of the language (related to Q2). We know that the study results are limited and do not provide statistical evidence to support general conclusions. However, we believe that it can be considered an initial step in planning future case studies. The validation reached its goal, i.e. the DSML has enough expressiveness to specify evaluation criteria at different universities.

The transformation *dscholar2Code* was validated using a proof of concept, to evaluate the coherence of the generated code in relation to the model input model specified using DSCHOLAR. To guide the transformation evaluation, the following research questions (RQ) were defined: RQ1: are all the evaluations specified in the model present in the component code? RQ2: are the evaluation criteria defined in the input model included in the component code? RQ3: are the mean grades correctly calculated? In RQ1 we want to know if all the evaluations were considered when generating the code and in

RQ2 we want to evaluate if the evaluation details (e.g. weight) were all mapped to code.

The validation was performed using models of three different universities. These models were specified in the case study carried out to validate the DSCHOLAR modeling language. At the end of the tests we observed that: (i) the code generator produced the code corresponding to the models in all the cases tested (related to RQ1); (ii) the grades calculated in our system were equal to the ones calculated in the university systems (related to RQ2 and RQ3). Based on these results, we conclude that, for the examples used, the transformation has covered all the evaluation criteria of the three universities, and is therefore satisfactory for the established purpose.

VII. CONCLUSIONS

In this project, we improved the development of Student Information Systems providing them with flexibility and productivity in modifying and evolving student evaluation criteria. The integration of SPL and MDD approaches was decisive in this improvement. In order to support it, we defined a small DSML, typical for each sufficiently complex variation point; the source code is partially manually generated using software frameworks, and partially automatically generated through transformations that generate code that integrates with the frameworks.

Our strategy is to evolve our SIS line iteratively, i.e. analyze variability points and, when adequate, provide them an MDD solution to improve development, as we did with the evaluation criteria component.

The solution allows a domain expert to model the evaluation criteria, using the small, easy to use DSML DSCHOLAR. Some training may be necessary for teachers or other non-IT professionals to use it, but the language has very few modeling elements and relationships types, which should facilitate its use.

Automatic code generation was achieved with the *dscholar2Code* transformation, but only to the point where C# source code is automatically generated. To achieve full automation, a mechanism that compiles and deploys the code in the production environment must be created and integrated with a configuration management mechanism that appropriately stores and selects the right binary code in each situation.

DSCHOLAR provides a way to identify each set of specific evaluation criteria, by identifying the *Entity* that uses these evaluation criteria. As a result, each professional authorized to define and to apply the evaluation criteria must be associated to a specific *Entity*. The Entity can be as encompassing as an entire University or as granular as a single teacher or class.

DSCHOLAR and *dscholar2Code* have been tested in case studies and a proof of concept and, although it has been demonstrated to be satisfactory, it has limitations. We are, however, working on a case study with professionals from several other universities to more accurately assess the solution and reach more generalized conclusions.

This project will evolve to cover other variation points of the SIS product family and to implement the build/deploy/configure mechanisms outlined in this section.

REFERENCES

- [1] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley Professional, 2000.
- [2] M. Brambilla, J. Cabot and M. Wimmer, *Model-Driven Software Engineering in Practice*, Morgan & Claypool, 2012.
- [3] E. S. Almeida, C.R.U.I.S.E: Component Reuse in Software Engineering, 1st ed., C.E.S.A.R e-book, 2007.
- [4] . A. Kleppe, J. Warmer and W. Bast, *MDA Explained: The Model Driven Architecture : Practice and Promise*, Addison Wesley, 2003.
- [5] F. Al-Hawari, A. Alufeishat, M. Alshawabkeh, H. Barham and M. Hababbeh, "The Software Engineering of a Three-Tier Web-Based Student Information System (MyGJU)," *Computer Applications in Engineering Education*, vol. 25, n° 2, pp. 242-263, March 2017.
- [6] Z. Molnár, D. Balasubramanian and Á. Lédeczi, "An Introduction to the Generic Modeling Environment," em *Model-Driven Development Tool Implementers Forum*, 2007.
- [7] Microsoft Corporation, "Overview of Domain-Specific Language Tools," 2016.
- [8] J.-P. Tolvanen and S. Kelly, "Model-Driven Development Challenges and Solutions - Experiences with Domain-Specific Modelling in Industry," *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pp. 711-719, 2016.
- [9] C. Consel and R. Marlet, "Architecture Software Using A Methodology for Language Development," em *Principles of Declarative Programming, 10th International Symposium, PLILP'98 Held Jointly with the 7th International Conference, ALP'98, Pisa*, 1998.
- [10] J. Greenfield , K. Short, S. Cook and S. Kent, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley, 2004.
- [11] S. Cook, G. Jones, K. Stuart and A. C. Wills, *Domain-Specific Development with Visual Studio DSL Tools*, Addison-Wesley Professional, 2007.
- [12] E. Visser, "WebDSL: A Case Study in Domain-Specific Language Engineering," em *Generative and Transformational Techniques in Software Engineering II. GTTSE*, Berlin, 2007.
- [13] P. Sochos, M. Riebisch and I. Philippow, "The Feature-Architecture Mapping (FARm) Method for Feature-Oriented Development of Software Product Lines," 2006.
- [14] J. González-Huerta, E. Insfran, S. Abrahão and J. McGregor, "Architecture derivation in product line development through model transformations," 2014.
- [15] N. Lahiani and D. Bennouar, "On the use of model transformation for the automation of product derivation process in SPL," 8 2018. [Online]. Available: https://www.researchgate.net/publication/327269080_On_the_use_of_model_transformation_for_the_automation_of_product_derivation_process_in_SPL.
- [16] A. P. Magalhães, R. S. P. Maciel and A. M. Andrade, "Towards a Metamodel Design Methodology: Experiences from a model transformation metamodel design," *27th International Conference on Software Engineering and Knowledge Engineering*, pp. 625-630, 2015.
- [17] C. Wohlin and A. Aurum, "Towards a decision-making structure for selecting a research design in empirical software Engineering," *Empirical Software Engineering - Springer*, 2014.
- [18] F. Buschmann, R. Meunier, . H. Rohnert, P. Sommerlad and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*, 1st ed., Wiley, 1996.

Towards Detecting and Managing Information Anxiety in the ICT Industry

Mark Micallef

Department of Computer Science
University of Malta
mark.micallef@um.edu.mt

Chris Porter

Department of Computer Information Systems
University of Malta
chris.porter@um.edu.mt

Abstract—Information Anxiety is defined as “stress caused by the inability to access, understand, or make use of information necessary for employees to do their job”. Even though it is in itself an intangible phenomenon, it is widely acknowledged and has been linked to impaired decision-making ability, information withdrawal, information avoidance, burnout and other health issues. The ICT industry is acknowledged to be a fully fledged knowledge industry. That is to say that its workers are mainly tasked with creating, understanding, applying and distributing knowledge as part of their day-to-day job. The industry is also characterised by disruptive innovations, continuously changing technologies and customers who constantly change their mind about what they want systems to do.

In this paper, we present the results of a study which tracked 18 participants for a period of one month in order to investigate the presence of information anxiety in the Maltese ICT industry. Our results indicate that information anxiety is present in non-trivial levels amongst our cohort of participants, with information overload being the predominant cause. Also, participants working in a quality assurance (QA) function are more exposed to the phenomenon as well as being exposed to a wider variety of sources of anxiety than developers. Experience is also shown to be a factor with participants being less prone to symptoms of information overload as they gain more experience in the field.

Index Terms—Information Anxiety, Software Engineering, Empirical Study

I. INTRODUCTION

In 1956, white-collar workers outnumbered blue-collar workers for the first time in US History [1]. Since then, a substantial portion of the commercial world (especially in western economies) has transitioned to the so-called “knowledge economy”. Even though balance sheets list bank balances and material assets, it is widely acknowledged that knowledge is now the primary commodity and knowledge flows are regarded as the most important factors in the economy [2]. Knowledge workers are employees who deliver value to an organisation through the creation, storage, retrieval, dissemination and most importantly, application of knowledge. Yet, where as manual labourers tend to have very well defined and routine jobs with known inputs and specific outputs, knowledge workers have a more dynamic and less tangible context to deal with.

The ICT industry is a prime example of a knowledge industry, composed exclusively of knowledge workers. A

typical software engineer is expected to possess up-to-date in-depth technical knowledge as well as intimate knowledge of whatever domain she happens to be working in, so as to be able to deliver technical solutions in that domain. These technical solutions tend to be highly complex in nature and constantly evolve so as to (1) help customers remain competitive in dynamic markets; and (2) remain current with latest developments and best practices in technology. In the course of a working day, a software engineer will participate in several knowledge-centred activities. These include attending meetings in which status updates are communicated, discussing solutions to technical problems with peers, implementing technical solutions, responding to customer and management queries, fixing issues, mentoring peers, and so on. This can become overwhelming. Misra and Stokols [3] lament the rapid growth and transmission of information in the digital age and argue that this poses new challenges for individuals dealing with the onslaught of communications from multiple sources. Unless properly managed, these challenges can lead to anxiety and burnout, which in turn leads to employees losing interest, exhibiting lower activity and feeling powerless [4]. This is where the phenomenon of *information anxiety* becomes more tangible.

Wurman [5] defined *Information Anxiety* as being the “stress caused by the inability to access, understand, or make use of information necessary for employees to do their job”. Considering the contribution that the ICT industry makes to national economies [6], as well as its key role in helping other industries to grow and flourish [7], we argue that the industry needs to be aware of information anxiety, be able to detect it, and subsequently manage it.

A. Research Questions

Whilst talk of employees burning out and leaving companies can be heard around many water coolers and in the halls of conference coffee breaks, to date and to the best of our knowledge, there has been no empirical study that demonstrates the presence of information anxiety in the ICT industry. In this paper, we begin to bridge this gap by following 18 ICT professionals in Malta for a period of one month in order to investigate the following research questions:

DOI reference number: 10.18293/SEKE2019-095

- RQ1: To what extent is information anxiety exhibited by employees in the Maltese ICT Industry?
- RQ2: What insights can be gained regarding the sources of information anxiety in the Maltese ICT industry?

II. INFORMATION ANXIETY

The term *information anxiety* was first coined by Wurman three decades ago, and was defined as “stress caused by the inability to access, understand, or make use of information necessary for employees to do their job” [5]. Since then, information anxiety has been studied in a number of contexts and from a variety of points of view. A substantial portion of the literature equates information anxiety with information overload. Hartog [8] sets out a whole set of lingo that has evolved around the concept including terms such as *info-glut*, *techno-stress*, and *information addiction*. Burkhardt et al. [9] define information anxiety as “a feeling of being overwhelmed that comes when confronting a large information task”. Using more visual language, McCarthy [10] defines it as “a kind of stupor, a feeling that we simply can’t keep up, can’t read fast enough, don’t know how to locate the information we need, don’t have time to sort through or think about all the data surrounding us”.

A. Sources of Information Anxiety

Whilst the tendency in the literature is to equate information anxiety with information overload, we argue that knowledge needs in the ICT are so complex that it is worth considering Wurman’s [5] wider definition and his explanation about the causes of the phenomenon. Wurman states that information anxiety can be caused by one or more of five possible causes: (1) an individual’s *inability to understand* the information required to carry out a task; (2) *feeling overwhelmed by the amount of information* that needs to be understood in order to complete a task (information overload); (3) someone not knowing whether the information required to complete a task *exists or not*; (4) knowing that information exists but not knowing *where to find it*; and finally (5) knowing where the information is but *not having access to it*.

B. Effects of Information Anxiety

Several studies have been carried out with the intent of uncovering the effect of information anxiety on employees. The three most common themes to emerge are (1) an impact on job performance; (2) psychological impact; and (3) a detrimental effect on the health of individuals.

1) *Impact on Job Performance*: After studying managers in a cross-national survey, Waddington [11] reports that 43% of managers claimed that information anxiety impaired their decision-making ability. A decade later, Williams [12] reported similar findings on impaired judgement in a similar study. Bawden and Robinson [13] claim that damage-limitation strategies resorted to by employees who were exposed to information anxiety tend to reduce their effectiveness and result in a negative impact on their employing organisation. Misra and Stokols [3] studied 484 undergraduate students

and found that those exposed to high levels of information anxiety devoted less time to contemplative activities. In the context of a knowledge organisation, this would result in employees not investing the time in innovation that is essential for maintaining a competitive edge.

2) *Psychological Impact*: Katopol reports that individuals exposed to information anxiety can have any combination of sensations including feeling overwhelmed, intimidated, fearful, lost, threatened, stressed, uncomfortable and/or timid [14]. Fox reports that the phenomenon leaves employees feeling a sense of loss of control [15] whilst Waddington [11] found a link to tension and reduced job satisfaction.

C. Detrimental effect on Health

In more severe and prolonged cases, information anxiety has been observed to affect individual’s physical health. Misra and Stokols [3] positively correlated higher levels of perceived information anxiety with incidences of physical health problems. Waddington [11] also reports that 42% of managers in his study believe that information overload affects their physical health. Similar conclusions were reached by Ifijeh [16], as well as Bawden and Robinson [13].

D. Measuring Information Anxiety

The main stumbling block with addressing information anxiety is the fact that it is difficult to detect before it reaches harmful levels. A review of the literature revealed two alternatives for measuring the levels of information anxiety in individuals.

Based on the argument in health research communities that the impact of objectively stressful events depends largely on one’s perception of their stressfulness [17], Cohen et al. [18] propose their Perceived Stress Scale (PSS). The method involves asking participants to answer 14 questions with one of *never*, *almost never*, *sometimes*, *fairly often* or *very often*. The answers are assigned numerical values of 0 through 4 for questions with a negative bias¹ and 4 through 0 for questions with a positive bias². The PSS score is calculated by summing the scores of all 14 questions and through three case studies, Cohen et al. showed the measure to be a reliable and sound predictor of various psychological indicators such as depressive symptomatology and social anxiety.

In order to assess information anxiety in the Canadian Air Force, Girard and Allison [19] used Wurman’s [5] five causes of information anxiety as a starting point. Participants in their three studies rated their agreement with statements on a scale of 1 (strongly disagree) to 5 (strongly agree). The statements were as follows:

- 1) I would not understand the data required to complete this task.
- 2) I would feel overwhelmed by the amount of data to be understood to complete this task.

¹Example: In the last month, how often have you felt nervous or stressed?

²Example: In the last month, how often have you felt the you were on top of things?

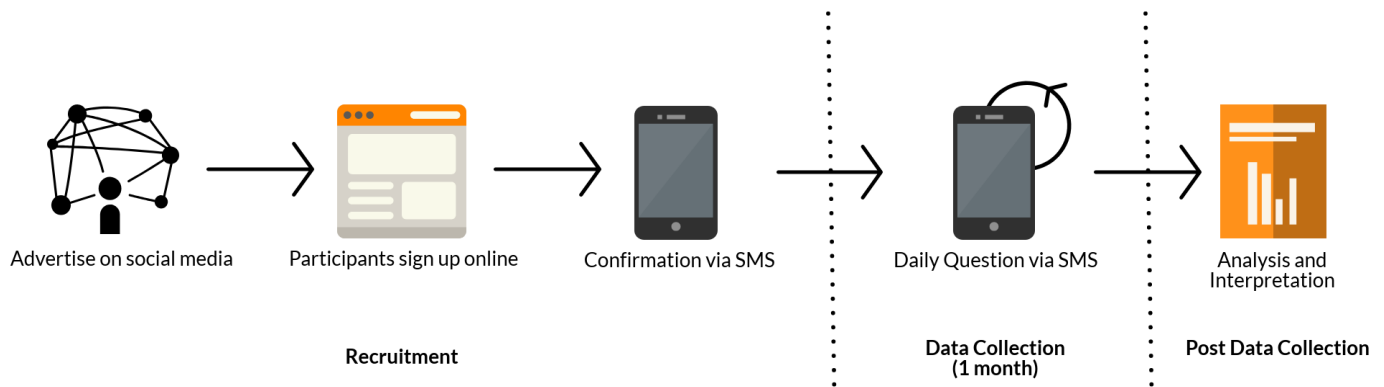


Fig. 1. An overview of the research protocol adopted for this study.

- 3) I would not know if certain data necessary for this task exists.
- 4) I would not know where to find data necessary for this task.
- 5) I would know exactly where to find the data, but I would not have the key to access it.

By summing up the ratings of individual components, Girard and Allison were able to measure the overall level of information anxiety experienced by an individual at a particular point in time. Although simple in nature, the measure is expressive both in terms of overall information anxiety, as well as in terms of the contribution of individual sources of anxiety.

III. METHODOLOGY

The research protocol was designed with the goal of regularly collecting empirical data from participants over a period of one month whilst minimising disruption to participants' daily routines. When considering the two alternatives for measuring information anxiety discussed in Section II-D, Cohen et al.'s approach [18] was ruled out because requiring participants to answer 14 questions on a regular basis would be disruptive. Instead, a modified version of Girard and Allison's [19] survey instrument was adopted. The method was aligned to the requirements of this study such that only one of the questions was asked to each participant every day. This reduced the disruption caused to participants whilst also collecting multiple data points as we followed individuals over a period of one month.

A survey mechanism utilising SMS technology was designed and implemented. As depicted in Figure 1, the study was partitioned into three phases. In the first phase, participants were recruited by means of adverts in social media groups that directed interested people to a sign-up form that collected some basic demographic information. They were later asked to confirm their participation via SMS so as to ensure that we were able to communicate with them over that medium.

The second phase consisted of data collection. Every working day for a month, participants received one question via

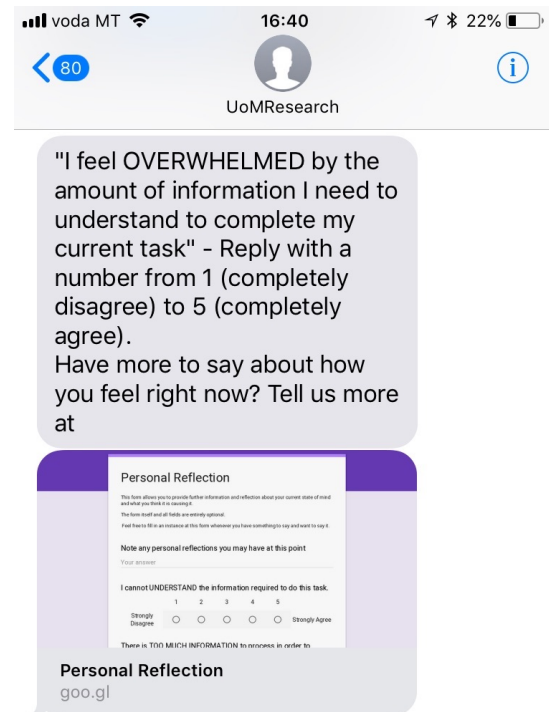


Fig. 2. A typical question sent to participants via SMS, including a link to the personal reflection form.

text message similar to the one shown in Figure 2 every day at 11am. The choice of question was randomised but once a question was asked, it would not be asked again before all other questions had been asked to the same participant. The time that questions were sent to participants was chosen deliberately so as to maximise the chance that participants have arrived at work, gone through their morning rituals and have started working. In order to minimise disruption, candidates were required to reply to each question by a text message containing a number between 1 (strongly disagree) and 5 (strongly agree). Participants were also provided with a link to a personal reflection form whereby they could optionally

provide further insight.

Finally, after the data collection period was over, the data was collated, analysed and interpreted.

IV. RESULTS

In this section, we present a digest of the key observations we gained from analysing the data collected during the study.

A. Participant Demographics

Eighteen participants signed up and participated in the study. Four demographic attributes were collected as information about participants: role, industry, experience and level of education. There was a relatively uniform distribution in all attributes except for education in which 11% of participants were educated to post-secondary level, 78% to undergraduate level and 11% to postgraduate levels. Participants were highly responsive during the study when it came to SMS responses, with a mean response rate of 79%. However, no participants availed themselves of the online reflection form to provide more unstructured information.

B. Initial Analysis

An initial analysis of the data revealed that when one analyses the cohort as a whole, only 17% of responses indicated levels of information anxiety. This was interesting to us as it went contrary to our initial hypothesis that information anxiety is significant problem in the industry. However, as we dissected the data further to consider participant demographics and individual sources of information anxiety, a more comprehensive picture emerged.

When one considers individual sources of information anxiety, it turns out that eight participants (44%) had a mean rating higher than three for at least one source. All eight of these participants indicated that information overload was the main cause of anxiety, with five participants indicating at least one more cause.

C. The influence of job role

Figure 3 plots a box chart showing the distribution of ratings provided by participants for each source of information anxiety, as well as the combined ratings of all sources, split by job role. One can immediately notice that job role seems to have a determining influence on the overall levels of anxiety registered by individuals. When considering all five sources of anxiety collectively, QA analysts registered comparatively high levels of anxiety with a median value of 2.5 and an upper quartile rating of 4. This contrasts with an upper quartile rating of 2 and 3 for software engineers and project leaders respectively.

Software engineers exhibited arguably negligible ratings across all sources of information anxiety except for information overload. The median of 2.5 and upper quartile value of 4 here indicates that there is a significant perception amongst software engineers in our cohort that they have too much information to process and feel overwhelmed by it. Furthermore, both QA analysts and project leaders exhibit

higher levels of anxiety as a result of information overload with over 50% of ratings from QA analysts being at or above 3.

Consistent with the other two roles, the QA Analyst cohort exhibited was mostly affected by information overload. However, the same cohort also has significant complaints of participants struggling with understanding, being unable to locate information and not having access to information. Project leaders have exposure to a wider range of anxiety sources than developers but the levels of complaints for the upper quartile only ever go beyond 3 for information overload.

D. The influence of experience

The most inexperienced participants reported the least problems related to being unable to understand information related to do their job. The highest problem in this area surfaced amongst participants with 6-10 years of experience in the industry. Conversely, participants with 3-5 years of experience reported the highest levels of information anxiety with 39% of information overload ratings for this cohort being 4 or 5. The data indicates that as participants gain experienced, they suffer less from information overload.

E. The influence of industry

We also analysed data from the point of view of industries in which participants worked. However, except for a marginally higher level of anxiety being reported by participants in the Gaming industry, the data indicates that the domain in which participants work does not have an obvious influence on the levels of information anxiety experienced.

F. Threats to Validity

The study suffers from two main threats to validity. Firstly, the sample size of 18 is not necessarily representative of the industry as a whole. Secondly, the duration of the study (1 month) when compared to the lifetime of a typical project. Having said that, the study is, to the best of our knowledge, the first of its kind to shed light on the levels of information anxiety in the ICT industry and lessons learned here will be taken forward to larger and more longitudinal studies that are being planned by the authors.

V. DISCUSSION

In this section we discuss the results of the exercise in the context of the research questions specified in Section I-A.

A. Presence of Information Anxiety in the ICT Industry

Although the data does not indicate an outright epidemic when it comes to information anxiety amongst our participants, it does indicate that there is cause for concern. This can be better appreciated when one considers that results obtained using Gurrard and Allison's [19] formula for calculating information anxiety can be misleading if one does not consider that it is an aggregating formula. This is because, our data shows that an individual is most likely to be exposed to one source of information anxiety. There were cases where individuals were exposed to two or at most three sources at any point

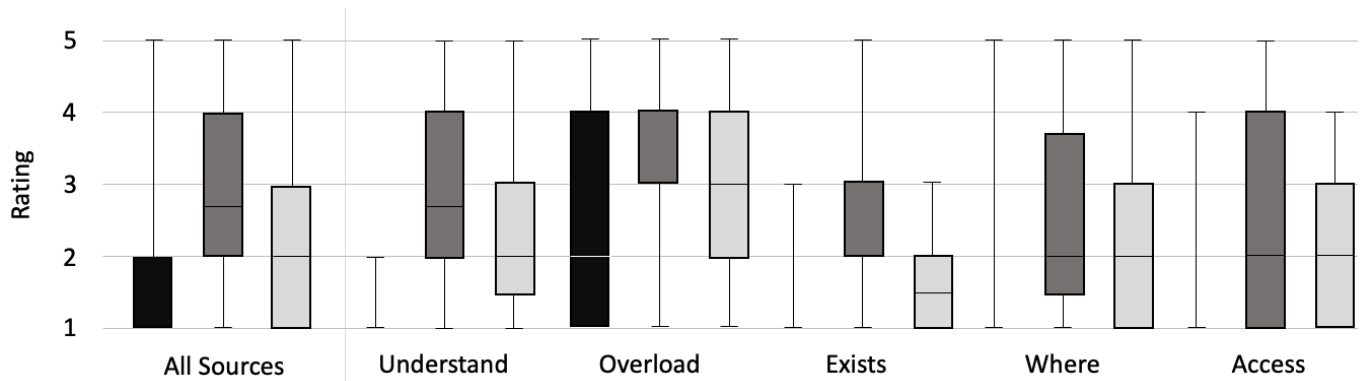


Fig. 3. Distribution of ratings for sources of information anxiety for Software Engineers (black), QA Analysts (dark grey) and Project Leaders (light grey).

in time but these were very rare. It is therefore likely that Girrard and Allison's formula will even out to a mean, as evidenced by their own studies where the mean value for anxiety readings hovered around 2.5. However, this does not mean that no information anxiety is present. Rather, we argue that a prolonged high rating for even one source of anxiety on one participant should be cause for concern and trigger an investigation to understand the circumstances leading to this.

Whilst it was out of scope for this particular study, individual employees ratings should also be examined over time to look for patterns that might be detrimental to employees psychological health. For example, regular short bursts of high anxiety would produce low average scores but employees could still have significantly negative experiences as a result.

Notwithstanding the small sample of participants, in answer to RQ1, the data collected for this study suggests that information anxiety is sufficiently present in the Maltese ICT industry to cause employers to devise ways of detecting it and measuring it.

B. Insights regarding the sources of Information Anxiety

The results from this study lead us to make three conclusions when it comes to the sources of information anxiety.

1) *The predominance of Information Overload:* Information overload was consistently shown to be the predominant cause of anxiety amongst our participants when compare to the other four sources regardless of job role, experience level or industry. This indicates that the literature's focus on information overload is well founded but not complete. For example, QA analysts demonstrated elevated levels of anxiety across multiple sources of anxiety. In such cases, whilst employees are still being suffering from anxiety the required solutions would be different to those required when the root cause is information overload.

2) *QA Analysts have an anxiety-inducing job:* The indication that QA Analysts suffer from higher levels of anxiety than other job roles is further explained when one examines the individual sub-components of information anxiety as experienced by people in this role. QA analysts tend to act as bridge communicators between the technical side of an organisation

(software engineers) and the so called product side of the business. The need to straddle two different worlds in which people often speak different languages and dialects. That is to say that whilst a software engineer might speak in terms of concepts such as algorithms, efficiency, load balancing and so on, a product-side employee would speak in terms of features, customers and other domain-specific concepts. QA analysts are responsible for ensuring that the technical side of the business delivers the right artefacts to the product side of the business and it is this straddling of two worlds that likely leads to elevated levels of anxiety.

3) *Experience is a factor:* One of the finding of the study is that symptoms of information overload decrease amongst participants with more experience. This is probably due to experienced individuals being able to develop soft skills for setting up barriers against information overload. For example, the simple act of saying No, I cannot deliver this extra feature in the next 2 weeks. has the effect of shielding an individual against any anxiety that might have resulted from committing to the extra task and setting himself up for failure. Individuals who have been around for longer are more capable of gauging the effect that a task might have of them and are more comfortable refuting that work.

VI. RELATED WORK

As discussed in Section I, to date and to the best of our knowledge, there has been no empirical study that demonstrates the presence of information anxiety in the ICT industry. The work the came closest to this is that of work that Mäntylä et al. [4], who did not explicitly discuss information anxiety but did discuss the ICT sector and highlight a number of related concepts and symptoms.

Mäntylä et al. claim that employees in the ICT sector are susceptible to "psychological diseases such as burnout, which lead developers to lose interest, exhibit lower activity and/or feel powerless" [4]. They propose a set of metrics called the VAD metrics, named after the first letter of three attributes which they measure: (V)alence, the level of enjoyment exhibited by individuals carrying out a task; (A)rousal, the level of alertness and readiness to act; and (D)ominance, the level to

which individuals are in control of their task. Using a database of 13,915 manually rated English words as created by Warriner [20], Mäntylä et al. calculated VAD scores on 700,000 Jira³ tickets containing over 2,000,000 comments. This resulted in a number of interesting conclusions regarding how valence, arousal and dominance in individuals varies based on how long a ticket remains open, how its priority changes and when it is eventually resolved. For example, their results indicate that burnout (low valence, low dominance and high arousal) is more likely when working on high priority issues that take a long time to resolve. Whilst Mäntylä et al. do not make the link with information anxiety, as discussed in Section I, software engineering is a knowledge-intensive activity and a prolonged unsolved issue is likely to be caused by a lack of understanding, information overload or issues with finding and/or accessing relevant information.

VII. CONCLUSION AND FUTURE WORK

In this paper we discussed the methodology and results of a study that was designed and executed to shed light on the presence of information anxiety in the ICT industry. By tracking anxiety levels amongst 18 participants on a daily basis for one month, we discovered that information anxiety is indeed present at non-trivial levels with information overload being the predominant cause. We also discovered that QA Analysts are more likely to suffer from the phenomenon, as are less experienced individuals. Finally, we discovered that the domain in which employees work does not have a significant bearing on anxiety levels.

A. Recommendations

This study contributes a non-intrusive data collection protocol that can help companies shed light on levels of information anxiety amongst their employees. Implementing a similar mechanism which collects data at possibly less regular intervals using company's intranet or time-sheet system would provide managers with valuable empirical data. This data can subsequently be analysed and tracked such that any emerging problems with information anxiety can be investigated and addressed before they cause substantial harm. Furthermore, the knowledge that employees in certain job roles are more susceptible to the phenomenon should lead to a review how these roles could be restructured in order to lessen the likelihood of information anxiety emerging.

B. Future Work

Having taken the first step towards investigating information anxiety in the ICT industry, we plan continuing this work across two prongs of research. number of related projects in the near future. Firstly, we would like to *set up a longitudinal study* which spans a longer period, ideally a year or more. This would allow us to gain insights into how and why anxiety levels change over time. We anticipate that the main

problem with such a study would be maintaining participant engagement, and to this end we are currently investigating the possibility of utilising gamification to achieve this.

Secondly, we would like to investigate the sources of anxiety in more detail. In particular, we are interested in understanding if there are any relationships between specific components and whether these are amplified in particular contexts. Insights gained in this area could lead future studies with a more refined focus on specific contexts or sources of anxiety.

REFERENCES

- [1] J. Kenway, E. Bullen, J. Fahey, and S. Robb, *Haunting the knowledge economy*. Routledge, 2006, vol. 6.
- [2] L. Chinho and T. Shu-Mei, "The implementation gaps for the knowledge management system," *Industrial Management & Data Systems*, vol. 105, no. 2, pp. 208–222, February 2005. [Online]. Available: <http://dx.doi.org/10.1108/02635570510583334>
- [3] S. Misra and D. Stokols, "Psychological and health outcomes of perceived information overload," *Environment and behavior*, vol. 44, no. 6, pp. 737–759, 2012.
- [4] M. Mäntylä, B. Adams, G. Destefanis, D. Graziotin, and M. Ortu, "Mining valence, arousal, and dominance: possibilities for detecting burnout and productivity?" in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 247–258.
- [5] R. S. Wurman, *Information Anxiety*. Doubleday, 1989. [Online]. Available: <http://books.google.com/books?id=f8ZoAAAAIAAJ>
- [6] M. K. Sein and G. Harindranath, "Conceptualizing the ict artifact: Toward understanding the role of ict in national development," *The Information Society*, vol. 20, no. 1, pp. 15–24, 2004.
- [7] D. Pilat and A. Wölfl, "Ict production and ict use: What role in aggregate productivity growth?" *The Economic impact of ICT-measurement, evidence, and implications*, pp. 85–104, 2004.
- [8] P. Hartog, "A generation of information anxiety: Refinements and recommendations," *The Christian Librarian*, vol. 60, no. 1, p. 8, 2017.
- [9] J. M. Burkhardt, M. C. MacDonald, and A. J. Rathemacher, *Teaching information literacy: 50 standards-based exercises for college students*. American Library Association, 2010.
- [10] M. McCarthy, "Mastering the information age," *Los Angeles: Jeremy P. Tarcher*, 1991.
- [11] P. Waddington, "Dying for information? a report on the effects of information overload in the uk and worldwide," *British library research and innovation report*, pp. 49–52, 1997.
- [12] C. J. Williams, *Reassessing the role of anxiety in information seeking*. University of North Texas, 2008.
- [13] D. Bawden and L. Robinson, "The dark side of information: overload, anxiety and other paradoxes and pathologies," *Journal of information science*, vol. 35, no. 2, pp. 180–191, 2009.
- [14] P. F. Katopol, "Information anxiety and african-american students in a graduate education program," *Education Libraries*, vol. 35, pp. 5–14, 2012.
- [15] J. Fox, "Conquering information anxiety. relief from your data glut starts here," 1998. [Online]. Available: <http://www.ibtpop.com/articles/dataglutrelief.doc>
- [16] G. Ifijeh, "Information explosion and university libraries: Current trends and strategies for intervention," *Chinese Librarianship: an International Electronic Journal*, 2010.
- [17] R. S. Lazarus, "Psychological stress and coping in adaptation and illness," *The International journal of psychiatry in medicine*, vol. 5, no. 4, pp. 321–333, 1974.
- [18] S. Cohen, T. Kamarck, and R. Mermelstein, "A global measure of perceived stress," *Journal of health and social behavior*, pp. 385–396, 1983.
- [19] J. Girard and M. Allison, "Information anxiety: Fact, fable or fallacy," *Electronic Journal of Knowledge Management*, vol. 6, no. 2, pp. 111–124, 2008.
- [20] A. B. Warriner, V. Kuperman, and M. Brysbaert, "Norms of valence, arousal, and dominance for 13,915 english lemmas," *Behavior research methods*, vol. 45, no. 4, pp. 1191–1207, 2013.

³Jira is a commonly used tool in the ICT industry which enables stakeholders to report issues with software (or request new features) and track work and discussions by developers whilst development progresses.

A Case Study of a Software Development Process Model for SIS-ASTROS

Camila Hübner Brondani, Otávio da Cruz Mello, Lisandra Manzoni Fontoura
Departamento de Computação Aplicada – DCOM
Universidade Federal de Santa Maria – UFSM
Santa Maria, Brazil
{chbrondani, odmello, lisandra}@inf.ufsm.br

Abstract — **Context:** technological innovation projects, developed in universities in partnership with companies and/or the government need processes that can handle the characteristics of the institutions involved. The academic environment is often used to dynamic methods, but contracts require plan-driven processes. **Goal:** the goal of this research is to understand the needs of the parties involved (university and government/enterprise) and provide an adapted software process to satisfy those necessities. **Method:** a case study considering a project between the Federal University of Santa Maria (UFSM) and the Brazilian Army (BA) for the development of an Integrated Simulation System was conducted. Initially, problems in the development were detected and a process was defined. It was then evaluated and improved over the iterations, through team meetings. **Results:** the experience acquired in the project was consolidated as lessons that could be used to assist the process definition of projects with similar characteristics. **Conclusion:** innovation projects involving the collaboration of universities, government and/or companies are successful if an adequate process is established to treat specificities of the academy, not only in relation to characteristics of the work but also the team.

Keywords—*process, triple helix, ASTROS, case study.*

I. INTRODUCTION

The university today, besides the academic activities and the pure research, promotes the development of applied research aiming to generate innovation solutions from issues presented by governmental institutions and enterprises. The triple helix thesis states that universities are distancing themselves from having a secondary social role, although important, of providing higher education and research, and is taking a primary role equivalent to the industry and the government, generating new industries and companies [1].

In this context, the Federal University of Santa Maria (UFSM) started a project to develop an Integrated Simulation System for the Brazilian Army (BA) in 2015. One of the initial challenges of the project was defining an adequate process model. Since there was an agreement between the BA and the UFSM with predetermined goals and deadlines, a plan-driven process model would be more satisfactory. On the other hand, the research needed for the system development could not be predictable, requiring investigation, prototype development and evaluation. The set of requirements was vague; the team was formed of high-skilled workers with autonomy. Based on these aspects, agile methodologies could be considered more proper.

Understanding the peculiarities of a project involving the university and government is crucial to choose an adequate process model that can satisfy the needs of both parties.

In this article, we will describe the lessons learned during four years of a research project between the UFSM and the BA and the process model that was used in this development, adapted over the iterations. Our goal is that the lessons learned and the process model can assist the definition and/or adaptation of models that are used in projects involving universities and governmental institutions or private companies.

The project in question proposes the development of a Tactical Virtual Simulator aiming the military training in tactical operations related to the use of an ASTROS battery (Artillery Saturation Rocket System).

Some important characteristics of the project are: need of meeting semi-annual goals pre-defined at the start of the project, difficulty of defining requirements due to the system's complexity, the unfamiliarity of the development team with the area of application, high-skilled workers, and constant need of innovative solutions research.

Those attributes led the definition of the software process. An evolutionary and iterative life cycle was defined, where intermediary versions of the software are generated and evaluated constantly by the client, easing the definition of new requirements for the next iteration. Besides that, milestones were defined with the purpose to satisfy contractual obligations. The process was evaluated and improved over the course of the project and the experience acquired was consolidated as learned lessons to be used in the development of other similar software projects.

This article is organized as follows: in Section II, important concepts to the comprehension of this work are introduced. In Section III, related works are discussed. In Section IV the context of the case study is described. In Section V, we define the proposed process. In Section VI, discussion and analysis are presented. At last, in Section VII we discuss our final considerations and comment on future works.

II. BACKGROUND

Modeling of software process has been a very challenging problem and constantly debated in the software development community in the past 30+ years, largely due to the complex nature of the software development process that involves not only the technical knowledge and skills but also

many other factors, such as human, management, quality assessment, and cost [2]. The modeling of business processes aids in the comprehension and optimization of existing business processes, and also in the conception of new business processes to make organizations more competitive and efficient [3].

Software development strategies have gradually shifted from the traditional waterfall model to more dynamic and responsive iterative, multi-cycle strategies. The reason usually cited is the need to minimize risk in the process [4].

Traditional iterative software development efforts such as spiral development or iterative enhancement can be considered adaptations of the waterfall software life cycle [5]. This is because these methods generally assume that the entire documentation required by the waterfall method will still be produced, but will be rewritten and updated during each cycle rather than once for the entire software process [5].

Agile processes are different from traditional software processes in that the time per cycle is very short and many fewer formal methods are employed. They focused on repeated lightweight practices for rapid and continuous delivery of software in small chunks with close collaboration from the customer as well as among members of the development teams.

No rigid plan or requirement is determined in advance, as these can change during the development process. Being flexible and adaptive to changes are in the DNA of agile methods while still achieve the ultimate goal of producing customer satisfied software within the time and cost framework [5]. Extreme Programming and Scrum are two software development processes that fit this description [5].

Many software development methodologies fall in between plan-driven development and agile development, and exhibit several of the characteristics of agile development. Examples include incremental development, prototyping, and DSDM (Dynamic Systems Development Method) [6].

To mitigate the impacts of abrupt paradigm changes and support organizations that don't want to stop following all traditional practices some proposals were developed for hybrid processes that incorporate principles of agile and traditional paradigms [7].

III. RELATED WORK

The study from Cotugno and Messina [8] presents an overview of the development process, focusing on the Scrum methodology adopted by the Italian Army for the development of software systems using open code technologies.

Benedicenti et al. [9] relate the experience of an agile application in the defense sector. They describe the experience of creating a control and command system for the Italian Army. The delivery of the project happened after 13 sprints of five weeks, meeting all the needs of the users and satisfying the regulatory requirements of the army. Acquiring this positive result demanded collective effort to change the development culture, since there was natural resistance to change, and the need of highest possible support level to

guarantee the continuity of the selected process. The article presents the positive results quantified.

As well as this article, the work from Cotugno and Messina [8] and Benedicenti et al. [9] describe methodologies and techniques used in the software development in an military environment. The main difference is that both are only focused on agile methods.

The work from Jenkis [10] describes the implementation experience of PRO-SOFTWARE, a software quality project involving the government, industry and academy (the triple helix). The goal was strengthening the software industry in Costa Rica, assisting organizations in improving their software processes. Therefore, Jenkis [10] proposes a methodology based on the quality improvement using the Capability Maturity Model (CMM) as base.

IV. CASE STUDY DESIGN

In order to address the research objective, we designed an exploratory case study, which involved a real-world software project. We define a software process based on identified process and analyze this process over several iterations. This section describes the design of the case study.

A. Project Context

The SIS-ASTROS project started in 2015, and is predicted to end in 2020. The main goal of the project is the development of an integrated simulation system to support the teaching of doctrines related to the use of a rocket artillery battery. The development team is formed of 7 doctor professors, 3 researchers, 4 developers, 7 master's degree students and 13 undergraduate students.

In addition, the requirement of the projects were described in high level of abstraction, the UFSM's team did not have the knowledge of the domain and the project required some innovative solutions, mainly related to the simulator's integrity, 3D scenarios generation and autonomous navigation. It is predicted to transfer the technology to the BA at the end of the project.

On the other side, the professors and researchers have long experience in the research field, developing researches to provide innovative solutions in different areas of computer science.

B. Methodology

At the beginning of the development, there was not a process model clearly defined in the project, so the artifacts were not standardized and the flow of activities did not follow a pre-defined roadmap. This scenario brought difficulties in the project management and fomented the definition and elaboration of a software process for the project. Therefore, from this necessity, this research project was initiated. The methodology used by the team to conduct the case study was composed of the four phases described below.

Diagnosis: identification of the problems happening on the project and possible solutions. In this stage, many problems related to the inexistence of a defined software process were found. The discovery of the problems occurred through meetings with the parties involved in the project.

Planning: from the problems identified in the last stage, a process model to be used in the project was proposed, aiming to solve these problems and satisfy the characteristics and necessities of the project and the team at the same time.

Implementation and Evaluation: during the three following years, the process was implemented and improvements were incorporated to it, intending to adapt the project to current needs.

Analysis: the results obtained over the course of the project are presented and the acquired experience is described as lessons learned.

C. Problems Diagnosis

The issues found during the Diagnosis phase can be summarized as follows.

Unfamiliarity with the application domain

The UFSM team did not have knowledge about military doctrines neither terminologies of the field. The manuals were rich in details and very extensive, making it difficult for the team to understand and learn.

Difficulties related to requirements definition

Being an innovative software, the set of requirements was not defined. There were a lot of concerns and doubts about how the simulation system would work and which features would be necessary.

Complexity of solutions

Complex and innovative computational solutions were required to solve the technical issues found during the development.

Rework

The team project was composed of workers with different skill levels, the professors and the researches were high skilled, master's degree students possessed an average level of skill and undergraduate students were low skilled. Since there was a large number of trainees, many problems in the source code were found, like defects, lines that were hard to comprehend and maintain, and issues related to class structuring.

Communication difficulties

Due to the hierarchic communication structure with the client, the information goes through several levels until the

decision taking. This communication flow causes problems like developmental delay, when for example, the team needs to wait for an answer to a doubt.

High team turnover

The students remain in the project while they are taking their graduation or master's degree course, on average two years. Therefore, we have high turnover.

Requirements instability/Changing Requirements

Constant changes in requirements, mainly during the test phases. Many changes occur because of divergent opinions, often due to lack of vision of the whole.

V. PROPOSED PROCESS OVERVIEW

In the planning phase, a software process was developed with intent of proposing solutions to the issues identified during the diagnosis phase while meeting the needs and peculiarities of the government and the academy. On one side, we have a stakeholder that gives priority to software documentation, rigid definition of iterations and deadlines, while on the other side, we have a self-managing team that is focused on development and coding.

The process initially defined was constantly evaluated through the phases and iterations (Implementation and Evaluation phase). The evaluations were performed during meetings, when the parties involved would discuss which practices gave positive results and which should be reviewed, and with this feedback, the process was improved.

The current process is described in Figure 1 (life cycle vision), Figure 2 (iteration activities) and Figure 3 (change management sub-process activities).

Some considerations about the process are described in the following section.

A. Process Life Cycle

Aiming to include the formal deliveries, foreseen in the contract, the life cycle was organized in phases and iterations, as depicted in Figure 1. Two phases are planned: initiation and construction, finished with a major milestone.

The initiation phase only happens once and is responsible for defining an overview (abstract) of the system in development and giving a clear comprehension of the business domain that is related to this system.

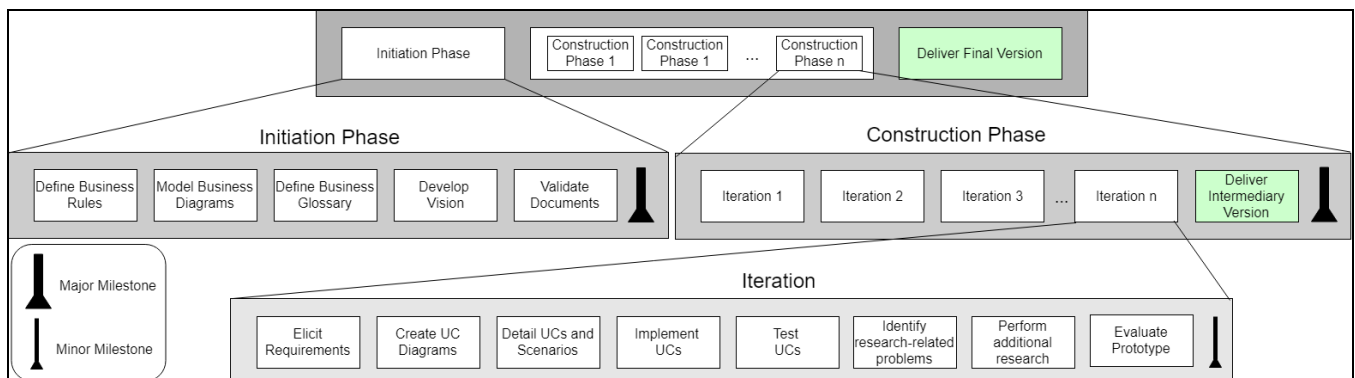


Fig. 1. Life Cycle Vision

The construction phase, on the other hand, is responsible for the execution of the technical activities that will generate a new version of the software. A project can have as many construction phases as needed, and each one can have multiple iterations. Both phases must respect contractual obligations, for this reason, they are finished with a major milestone that indicates a formal delivery to the client.

Since the phases usually refer to bigger time spans (semesters, years), it was chosen to break them in many iterations with the purpose of speeding up the process. Each iteration has its complete development cycle, from requirements definition to version evaluation (Figure 2). The software versions developed in the iteration are always delivered when the phase ends (major milestone). The number of phases and the amount of iterations in each of these phases depend on the project and can be adjusted.

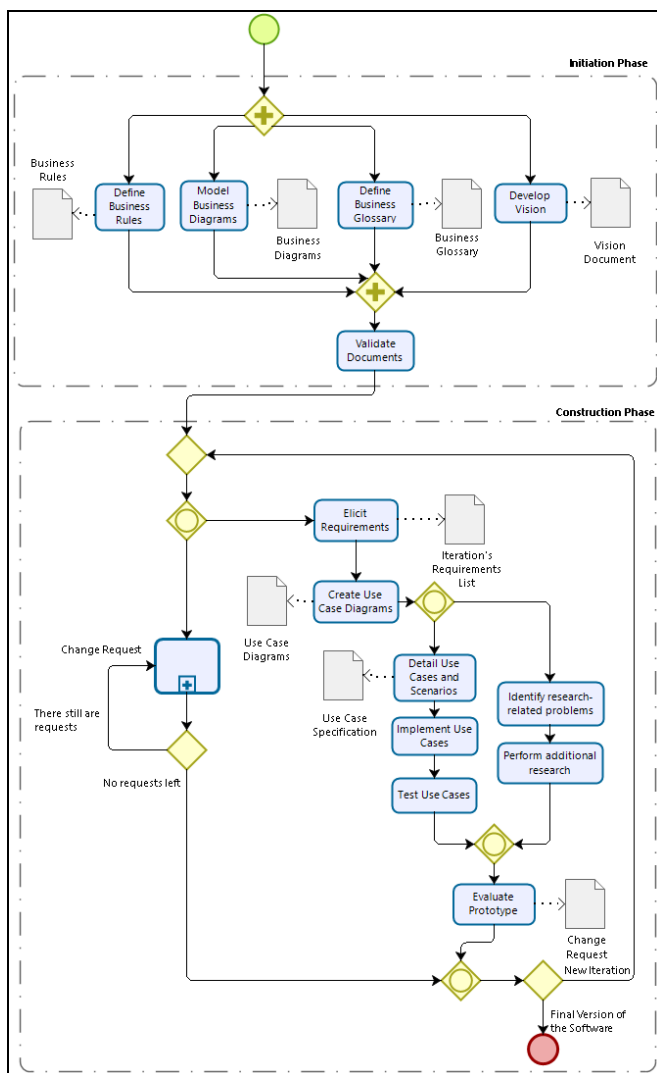


Fig. 2. Iteration Activities

At the end of the iteration, meetings with the client are held to present the intermediary version of the software, in which possible improvements, changes and evolutions are discussed. These meetings are important to track the current progress of the development team.

B. Activities and Artifacts

The initiation phase is composed of four main tasks that happen at the same time: define business rules, model

business diagrams, define business glossary and develop vision. These tasks generate the artifacts business rules, business diagrams, business glossary and vision document, which are formally evaluated by the client. Before the construction phase starts, it is extremely important that the artifacts generated during the initiation phase have been approved by all the stakeholders (task validate documents). When the respective documents are finished and approved, successive iterations start in each phase. Each one has a set of tasks that generate an intermediary version of the software. In the first task, the stakeholders meet to define the requirements that must be implemented in the cycle.

After the requirements of that iteration are defined and prioritized, the specification and detailing tasks start. Diagrams and requirements specification documents are created to assist the team members during the development and the technology transfer process. All the artifacts created in this phase are managed in a requirements management tool.

The tasks identify research-related problems and perform additional research are executed simultaneously, due to constant innovative solutions research. These are incorporated in the simulator in the next iteration.

Once the modeling ends, the team can finally start implementing and testing. If there are issues with a feature that cannot be fixed during the defined cycle, or the programmers are late in the development, an artifact is generated reporting the features that could not be finished, so they can be implemented in the next cycle. As soon as the iteration finishes, the client validates the intermediary version of the software, defining additions or changes that should take place. These are documented and serve as input for the next iteration's requirements definition.

Change requests can be submitted at any time, either to include or modify a requirement that was previously defined. In the main process, the procedure of submitting a change request is seen as a sub-process (Figure 3).

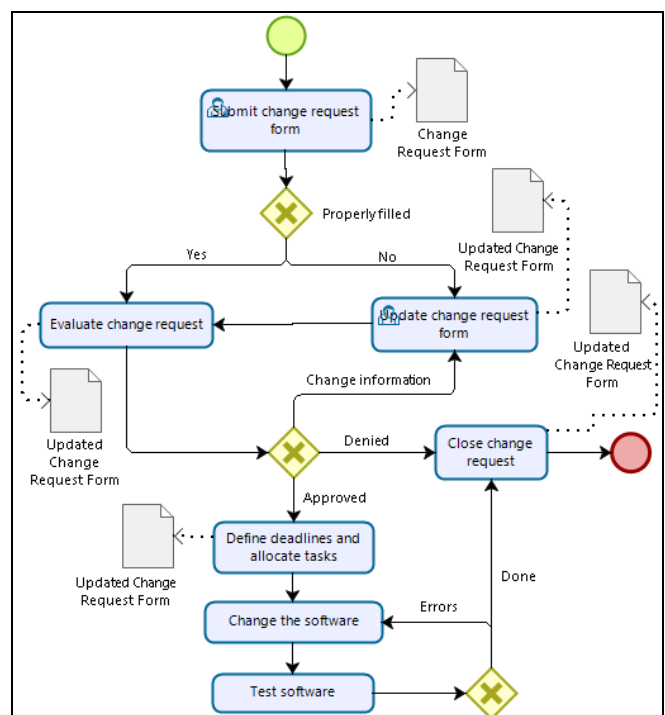


Fig. 3. Change Management Sub-process Activities

This sub-process is basically a flow of activities to manage the changes in the project. First, a stakeholder submits a change request, which is reviewed by a committee and, if the request is relevant, the change is incorporated in the version of the software. However, there are times when no requests are submitted in an iteration, so the change management process will not necessarily occur in the flow, therefore, being optional.

C. Roles

The project team was organized in levels: researcher professors (part time), professionals and researchers (full time), master's degree students and trainees (part time). The researcher professors guide the students in solving research problems and developing their academic works.

Professors are also responsible for the project management. Professionals and researchers are in charge of planning the tasks to be assigned to each member of the team, communicating obstacles to the management and organizing the daily routine of the team. Master's degree students are responsible for guiding the undergraduate students in their activities, helping solving issues regarding tasks assigned to them. The product owner is responsible for the communication between the development team and the client; all the requests from the team are centered on this person, which will track them until they are complete.

The team is collaborative, all the workers are assigned to close workrooms and there is constant exchange of knowledge between the team members.

VI. DISCUSSION AND ANALYSIS

The plan-driven approach served as foundation to the process definition. Using the basic principles: analysis, design, construction and verification, we have the basis for the flow of activities, supporting development of specific documents related to each phase of the project. The contractual aspect of the project, that demands deliveries on a timeframe, is contemplated with milestones at the end of each phase. The documents submitted are important for the requirement of technologic transference at the end of the project.

Allied to traditional models, we've decided to apply some characteristics of agile methodologies to the process as well, in order to emphasize the collaborative and communicative principles of the team and the final user, allowing incremental deliveries and also supporting the constant change requests without affecting or causing time and/or financial damage to the project [11].

Therefore, the method used in the creation of the process was defined as a hybrid between plan-driven and agile models, using the most advantageous characteristics, aligned with the goals of the project. In addition, for each issue found, actions were taken in the software process, aiming to solve or minimize them. They are described below.

Unfamiliarity with the application domain

The solution found was including some tasks at the start of the process with the purpose of comprehending the application domain. Diagrams that represented the domain were elaborated in collaboration with the stakeholders. Besides that, glossaries were also created, that are being

maintained through the course of the project. The BA team has been formally validating these documents.

Difficulties related to requirements definition

It was decided to work on intermediary versions of the software that were evaluated periodically by the BA team. When all the parties approved the prototype, a new set of requirements for the next iteration would be defined.

Complexity of solutions

It required applied research and development of master's essays and final papers exploring necessary solutions for the development of the simulator guided by a researcher in the field.

Rework

The solution was the constant refactoring of the source code, especially at the beginning of the project. Before the formal deliveries, there were periods intended for the code refactoring, with the purpose of improving legibility and documentation, as well as removing unnecessary code lines. We now focus on continuous source-code reviewing. Additionally, there is an internal hierarchy where experienced members assist new ones, helping them developing high-quality artifacts.

Communication difficulties

A formal communication flow was defined so that the parties involved track the information requests.

High team turnover

Some experienced professionals (researchers and programmers) were hired full time. Teamwork is encouraged and constant experience exchange between trainees and experienced members happen, thus, the team shares the knowledge of the system.

Requirements instability/Changing Requirements

Usage of incremental and iterative development, focused on periodic presentations of the intermediary version of the software. A formal change request process was also created.

Based on the results obtained by the execution of the process and the continuous monitoring of the team and the client since the beginning of the project up until now, it was possible to define some of the best practices and adopted decisions that reflected positively on the quality and progress of the project. It is believed that these practices can be applied in software development projects that involve academy and government and/or industry.

Client's periodic homologation

Iterative development allows the team to deliver a functional product to the client at the end of each iteration or cycle. The client can use this prototype over a period of time and provide feedback for the developers in terms of definition of new requirements, change requests and issue reporting. Usually, changes are incorporated into the requirements baseline to be implemented in the next iteration.

Use of diagrams to represent the business domain

Business diagrams helped the team to comprehend the business domain, making future communications more fluid. These diagrams were also used by the client to communicate

with other parties involved in the project. It was possible to represent the BA doctrines fully and clearly, preventing the team from reading manuals that are complex and difficult to understand.

Cooperative work

Team members can learn from each other. The more experienced guide the less experienced. Additionally, each team member knows what the others are developing, and can exchange information. The development of a particular activity becomes priority of the group as a whole, and not property of a certain team member only. The master's degree students mentor undergraduate students in research, that way, team members develop a common sense of responsibility that brings them closer.

Effective communication

The agile processes support the idea of face-to-face communication as the most effective and efficient method of transmitting information in the development team. The UFSM team is allocated in a sole environment. However, since the client team is located in another state, face-to-face communication is not possible. Therefore, to build an efficient communication method, it was necessary to center the communication on the Product Owner. This person is responsible for bringing the military vision to the project and evaluating, along with the team, the enhancements or changes that should take place to ensure that the software fulfill the needs of the BA. Bimestrial face-to-face meetings are scheduled.

Definition of a change management process

The change management process helped to monitor change requests and limited the number of unnecessary requests without the global comprehension of the system.

In the SIS-ASTROS project, we have defined one initiation phase and five construction phases, with duration of six months each. In each construction phase, three bimestrial iterations were established, since there are many part-time workers in the project that need to conciliate their work in the project with their academic obligations, teaching, in the case of professors, and classes and university assignments, in the case of students.

During this time, we managed to meet the goals defined in the project within the time and budget. The BA is satisfied with the results obtained and future projects are being discussed. The formal change request process reduced rework, and the amount of defects in the software has been dropping over time at the same pace performance (response time) has been increasing.

VII. CONCLUDING REMARKS

Projects involving the collaboration between universities and government and/or industry are successful if suitable procedures to handle the needs and peculiarities of the parties involved are established. In the described case study, hybrid process types proved to be satisfactory because they explore plan-driven characteristics – based in contracts, at the same time agile methods are suitable for innovative projects, which involve high-skilled professionals.

It was possible to experience practices from both investigated process in this project, reflecting positively in the developmental quality and progress, solving issues previously detected and establishing a set of learned lessons that can be used in other similar software development projects.

As future work, the main idea is to review the process periodically along with the team, continuously verifying the relevance of the activities and artifacts. As the process is thoroughly used, it may be possible to optimize some activities, thereby making the process less bureaucratic.

The fact that this approach was only applied in one project, even if in successive iterations during three years, was a limitation associated with this article.

ACKNOWLEDGMENT

We thank the Brazilian Army for the financial support through the SIS-ASTROS Project (813782/2014), developed in the context of the PEE-ASTROS 2020.

REFERENCES

- [1] H. Etzkowitz and C. Zhou, "Hélice Tríplice: inovação e empreendedorismo universidade-indústria-governo," *Estudos Avançados*, vol. 31, no. 90, pp. 23–48, 2017.
- [2] R. A. Haraty and G. Hu, "Software Process Models: A Review and Analysis," *International Journal of Engineering & Technology*, vol. 7, pp. 325–331, 2018.
- [3] K. C. Laudon and C. G. Traver, *Management Information Systems*, 12th ed. Sao Paulo: Prentice Hall, 2011.
- [4] H. M. Olague, L. H. Etzkorn, W. Li, and G. Cox, "Assessing Design Instability in Iterative (agile) Object-Oriented Projects," *Journal of Software Maintenance and Evolution: Research and Practice*, pp. 237–266, 2006.
- [5] B. Ramesh, L. Cao, and R. Baskerville, "Agile Requirements Engineering Practices and Challenges: An Empirical Study," *Information Systems Journal*, vol. 20, no. 5, pp. 449–480, 2010.
- [6] G. Van Waardenburg and H. Van Vliet, "When agile meets the enterprise," *Information and Software Technology*, vol. 55, no. 12, pp. 2154–2171, 2013.
- [7] W. Chaves, D. S. Carvalho, P. F. Rosa, S. Soares, M. Antonio, and L. C. Buiatte, "A comparative Analysis of the Agile and Traditional Software Development Processes Productivity," *30th International Conference of the Chilean Computer Science Society*, IEEE, pp. 74–82, 2012.
- [8] F. Cotugno and A. Messina, "Adapting SCRUM to the Italian Army: Methods and (Open) Tools," *IFIP International Federation for Information Processing*, 2016.
- [9] L. Benedicenti, A. Messina, and A. Sillitti, "iAgile: Mission Critical Military Software Development," *International Conference on High Performance Computing & Simulation iAgile*, pp. 545–552, 2017.
- [10] M. Jenkins, "PRO-SOFTWARE: A Government-Industry-Academia Partnership that Worked," *17th Conference on Software Engineering Education and Training (CSEET'04)*, 2004.
- [11] I. Sommerville, *Engenharia de Software*, 9th ed. São Paulo: Pearson Prentice Hall, 2011.

Experiences on applying SPL Engineering Techniques to Design a (Re) usable Ontology in the Energy Domain

Javier Cuenca, Felix Larrinaga

Electronics and Computing Department
Mondragon University/Faculty of Engineering
Mondragon, Spain

jcuenca@mondragon.edu, flarrinaga@mondragon.edu

Edward Curry

Insight Centre For Data Analytics
National University of Ireland
Galway, Ireland

edward.curry@insight-centre.org

Abstract—Global ontologies must provide a balance of reusability-usability to minimize the ontology reuse effort in different applications. To achieve this balance, ontology design methods focus on designing layered ontologies that classify into abstraction layers the common domain knowledge (reused by most applications) and the variant domain knowledge (reused by specific application types). This classification is performed from scratch by domain experts and ontology engineers. Hence, the design of reusable and usable ontologies that represent complex domains takes a lot of effort. Considering how common and variant software features are classified when designing Software Product Lines (SPLs), we argue that SPL engineering techniques can facilitate the domain knowledge classification taking as reference existing ontologies. In this paper, we show the experiences of applying SPL and ontology design techniques in combination to design a reusable and usable global ontology for the energy domain. Domain experts and ontology engineers evaluated the proposed method. The results show that SPL engineering techniques enable a systematic and accurate domain knowledge classification, thus saving ontology design effort.

Keywords—ontology design; ontology reusability; ontology usability; Software Product Line; energy domain.

I. INTRODUCTION

Ontologies are formal vocabularies that represent a data domain as a set of concepts and relations. The main ontology elements are classes (to represent entities, i.e., *device*, *appliance*), instances (individuals that belong to a certain class, i.e., *MU_fridge*, *MU_building_11*) and properties (relations that relate classes and individuals, i.e., *isA*, *isIn*) (Fig. 1). With these elements, ontologies enable to create a generic knowledge that can be shared across different software applications [1].

Among the different ontology types, *global ontologies* include common vocabularies to provide a common domain knowledge representation (i.e., Soupa [2]). The knowledge of global ontologies is a reference to develop ontologies for specific applications (*application ontologies*) [3]. This common knowledge representation overcomes the vocabulary differences and the heterogeneity of ontologies in the domain concerned to enable interoperability between ontology-based applications [3].

A global ontology must represent abstract knowledge to support different applications: it must be *reusable* [4]. However, if the ontology is too abstract, the effort of adapting it to satisfy

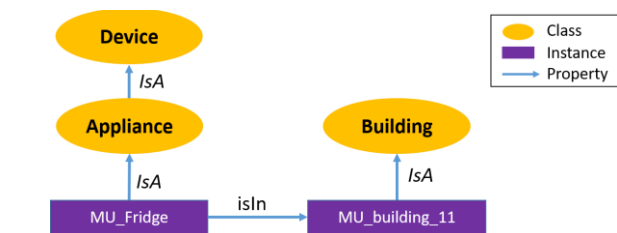


Figure 1: Ontology example

specific knowledge requirements would be high. Therefore, a global ontology must be as specific as possible to minimize the ontology reuse effort when it is reused to develop application ontologies: it must be *usable* [4]. Both reusability and usability are objectives in “*in natural conflict*” [4]. Hence, a global ontology must achieve a balance of reusability-usability so that it is reused in different applications with moderate effort.

To date, *layered ontologies* have been applied to achieve a balance of reusability-usability (i.e., OntoCape ontology [4]). They separate and classify into abstraction layers the *common domain knowledge* (reused by most applications) and the *variant domain knowledge* (reused by specific application types). We consider an *application type* a family of applications that perform similar tasks. In addition, the knowledge of each layer is divided into ontology modules, which represent the knowledge of a particular topic of the represented domain (each module imports the modules whose knowledge it requires or extends) [5]. Layered ontologies enable ontology developers to reuse only the necessary knowledge at the proper level of abstraction to develop application ontologies, thus reducing the ontology reuse effort in different applications [6].

Current methods applied to design reusable and usable ontologies [4], [7], [8] provide guidelines to define the ontology layers and the knowledge they include. However, they do not provide systematic guidelines to decide whether the domain knowledge is common or variant and in which layer it is placed. Domain experts and ontology engineers define and classify from scratch the common and variant domain knowledge. Ontologies are usually developed in complex domains (i.e., energy) [7]. Hence, a significant effort is required to classify the ontology knowledge from scratch by applying existing methodologies when designing reusable and usable ontologies.

Layered ontologies are quite similar in concept to Software

Product Lines (SPLs), software families that contain common reusable parts and variable parts that depend on specific customer needs [9]. To design SPLs, software features for a set of applications are analyzed and classified into *common features* (common to most applications) and *variant features* (implemented by specific applications) [9]. This process is known as *commonality and variability analysis* (CVA) [8]. This analysis is usually performed systematically taking as reference the software feature similarities and differences of legacy applications to complement domain experts' and software engineers' expertise [8]. This approach avoids classifying the software features from scratch, thus reducing the SPL design effort [10]. In addition, the software features reused by few applications are identified and the accuracy of the software feature classification is maximized [10].

After several decades of ontology development, many ontologies are available and developed to support certain application types. Hence, the CVA applied to design SPLs can be replicated in the ontology engineering field to design reusable and usable ontologies. The similarities and differences of the knowledge represented by existing ontologies can be analyzed. This analysis would complement domain experts and ontology engineers' expertise and prevent them from classifying the domain knowledge from scratch. In addition, the variant domain knowledge reused by specific applications could be identified, thus leading to an accurate domain knowledge classification.

In this paper, we show the experiences of applying SPL engineering techniques and ontology design principles in combination to design a reusable and usable global ontology for the energy domain. We discuss how applicable are SPL engineering techniques to design reusable and usable ontologies and the benefits they bring to the ontology design process.

The paper is structured as follows. Section 2 motivates the application of SPL engineering techniques to design a reusable and usable global energy ontology. Section 3 positions the proposed method with current ontology and SPL design methodologies. Section 4 describes the process we followed to design a global energy ontology by applying SPL engineering and ontology design techniques in combination. In Section 5, domain experts and ontology engineers evaluate the proposed method and Section 6 summarizes the learnt lessons. Section 7 summarizes the conclusions of the study and the future work.

II. MOTIVATIONAL SCENARIO

From the beginning of the current decade, ontologies that represent the knowledge from different energy domains have been developed. These ontologies support energy management applications focused on improving the current grid sustainability to make the Smart Grid vision a reality [11]. These applications can be classified into different types according to the Smart Grid scenario/infrastructure where they are deployed, i.e., Smart Home or building energy management applications. We define these application types as *Smart Grid scenarios*. Each Smart Grid scenario encompasses more specific application types. For instance, within Smart Home energy management applications, there are applications focused on home energy assessment or appliance Demand Response (DR) management. To see in more detail this classification we refer the reader to [12].

Energy ontologies are heterogeneous, since they represent the same energy domains (i.e., energy equipment data) with different vocabularies [12]. The energy management in real scenarios will require the knowledge exchange among applications that operate in different scenarios. This knowledge exchange is hampered by the heterogeneity of energy ontologies. Hence, there is the need to create a global ontology that provides a common energy domain representation [12]. It should support different energy management applications and provide balance of reusability-usability to minimize the ontology reuse effort in each application. Since the energy domains are complex, the application of existing reusable and usable ontology design methodologies would require a great effort. Since there are many developed energy ontologies, their knowledge similarities/differences can be analyzed to save ontology design effort.

III. RELATED WORK

The first knowledge classification proposals correspond to frameworks that classify ontologies according to their generality/specificity level. Guarino [13] presented the first ontology classification framework, which was refined by Gomez-Perez [14]. Layered ontologies (introduced in Section 1) are based on the aforementioned frameworks and they are the main approach to design ontologies that provide a balance of reusability-usability. In the last decade, several reusable and usable ontology design methodologies (based on the layered ontology approach) have been proposed. Spyns et al. [8] presented the DOGMA methodology, which specifies how to represent and separate the common and variant domain knowledge to design ontologies that provide a balance of reusability-usability. Thakker et al. [7] set out a methodology to develop reusable and usable ontologies for complex domains. Morbach et al. [4] developed the OntoCape ontology, a reusable and usable ontology for the chemical process engineering domain. In these methodologies, the classification of the domain knowledge is performed from scratch based on domain experts' and ontology engineers' expertise. They analyze the knowledge requirements of the application types that will be supported by the ontology (in collaboration with stakeholders). In contrast, in the method presented in this paper the common and variant domain knowledge is identified and classified through a CVA of existing ontologies conducted by applying SPL engineering techniques. Apart from this differential aspect, the proposed method applies the ontology design principles applied by current methodologies.

Regarding SPL design approaches, Pohl et al. [9] provide guidelines and enumerate the techniques to conduct a CVA. The proposed method follows these guidelines. In addition, several works have combined techniques from ontology and SPL engineering. Most of these works consist on the use of ontologies to improve the representation of common and variant software features of SPLs [15], [16]. In other works, ontologies and SPLs have been applied in combination, i.e., to manage cloud service configurations [17]. The proposed method also combines ontology and SPL techniques. In this case, SPL engineering techniques are applied to improve the ontology design process.

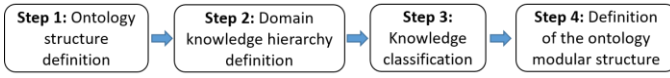


Figure 2: DABGEO design steps

IV. PROPOSED METHOD

This section explains the process we followed to design a global energy ontology: the DABGEO ontology. DABGEO provides a common energy domain representation and classifies the domain knowledge into abstraction layers to provide a balance of reusability-usability. DABGEO and its documentation are published online¹, so that ontology developers can understand its structure and reuse it. The DABGEO design and development team included energy domain experts and ontology engineers. DABGEO was designed by applying SPL engineering and ontology design techniques in combination. The design process followed four steps (Fig. 2), described in the following subsections.

A. Step 1: Ontology Structure Definition

In this step, the DABGEO structure was defined by the domain experts based on the layers proposed by the methods reviewed in Section 3. DABGEO includes three layers (Fig. 3). The *common-domain layer* represents the knowledge common to the Smart Grid scenarios. Variant domain knowledge still common to more than one Smart Grid scenario is included in the *variant-domain layer*. The *domain-task layer* includes the knowledge reused in specific Smart Grid scenarios and is divided into the *Smart Grid scenario* and the *application type* sublayers. The former represents the knowledge reused by a certain Smart Grid scenario and the latter represents the knowledge reused by a certain application type of a Smart Grid scenario. The lower the layer, the more specific the knowledge it represents. Hence, the modules from low-level layers will import the modules from upper layers. For more information about the ontology structure, we refer the reader to the ontology publication page¹.

B. Step 2: Domain Knowledge Hierarchy Definition

In this step, domain experts and ontology engineers defined and structured DABGEO knowledge. The knowledge was defined as a knowledge hierarchy where the represented domains were divided into specific knowledge pieces. This knowledge hierarchy enabled (1) to separate the abstract knowledge that is likely to be reused in most of applications from the specific knowledge and (2) to classify of the defined knowledge pieces into the layers of the ontology structure (performed in Step 3). Fig. 4 shows part of DABGEO knowledge hierarchy, which includes three elements:

- *Domains*: the domains represented by the ontology are located in the first level of the hierarchy. For instance,

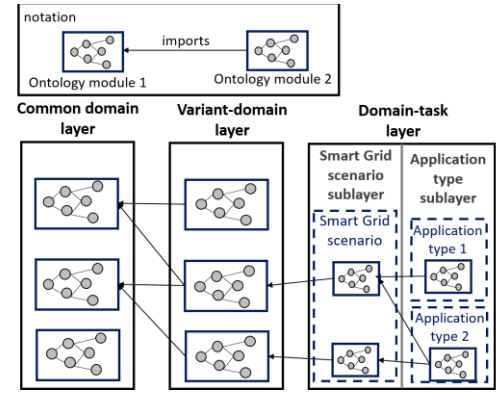


Figure 3: DABGEO ontology structure

the *energy equipment domain* encompasses the knowledge about energy devices and their operation.

- *Subdomains*: they cover the knowledge of an important part of the domain and are located in the second level of the hierarchy. For instance, the *energy equipment domain* encompasses the *energy consumption systems* and *device operation* subdomains, which represent the knowledge about energy consumption devices and device functional features respectively.
- *Knowledge Areas (KAs)*: in the third level of the hierarchy, consider a KA as a potential module of the designed ontology that encompasses the knowledge of a specific topic of a subdomain. For instance, within the *energy consumption systems subdomain* the *appliances KA* represents the knowledge about appliance types. KA can be divided into “child” sub-KAs that represent more specific knowledge. For example, the *appliances KA* includes the *white goods* and *brown goods* KAs, which represent the knowledge about white and brown goods types respectively. Hence, a sub-KA extends the knowledge of a “parent” KA. Finally, some KAs may require the knowledge from other KAs to represent the knowledge they encompass. For instance, the *energy consumption systems operation KA* describes the states and functionalities of energy consumption systems. It requires the knowledge of *device state* and *device functionality* KAs, which represent the knowledge about device states and functionalities respectively.

The proposed method classifies the ontology domain knowledge based on a CVA of existing energy ontologies. Thus, the knowledge hierarchy includes the knowledge represented by existing ontologies. The domain experts and ontology engineers collaborated to perform a manual analysis of the elements of existing ontologies in the Protégé ontology editor² to identify the domains they represent and to divide them into KAs. The identified domains were divided into subdomains, which were divided into KAs taking as reference the Competency Questions (CQs) answered by existing ontologies. CQs are the queries that ontologies should answer to ontology-based applications, and they are used to define the ontology functional

¹ <http://www.purl.org/dabgeo>

² <https://protege.stanford.edu/>

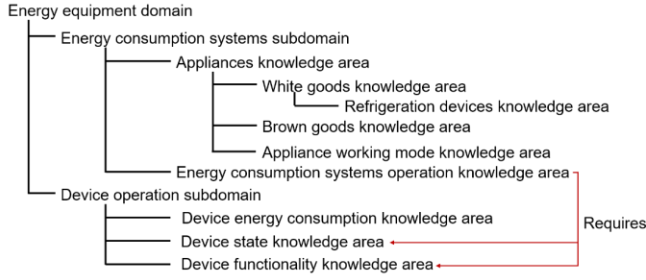


Figure 4: Part of the knowledge hierarchy of DABGEO

requirements [18]. To answer each CQ the ontology must include a specific part of the represented knowledge [18]. Thus, CQs are a natural guide for splitting the ontology knowledge into KAs [19]. CQs offer an abstract method to divide the knowledge represented by existing ontologies regardless of their heterogeneity. However, the CQs defined to develop ontologies are not always available [19].

Therefore, ontology engineers manually analyzed the elements of existing energy ontologies (classes and properties) to identify and extract the CQs they answer. This strategy is also followed when designing SPL taking as reference existing applications [20]. For instance, the existing energy ontologies include the *consumesEnergy*, *actuallyConsumesEnergy* and *maxConsumesEnergy* properties to answer the *What is the energy consumption of a device?*, *How much energy is a device consuming?* and *What is the maximum energy consumption of a device?* CQs respectively. To avoid an unmanageable number of KAs, the CQs covering similar topics were grouped by domain experts to define a KA that encompasses all the knowledge required to answer grouped CQs. For instance, the aforementioned CQs describe knowledge about device energy consumption. They were grouped into the *device energy consumption KA* (it also includes CQs answered by other energy ontologies), which encompasses the knowledge that answers these CQs. The defined KAs were classified into domains and subdomains according to the knowledge they represent and into a hierarchy level according to the knowledge they require or extend.

Finally, the domain experts provided a complete description of each KA and the knowledge it encompasses.

C. Step 3: Knowledge Classification

In this step, the ontology engineers classified each defined KA into one layer by applying SPL engineering techniques. First, the existing energy ontologies were analysed manually with Protégé to determine whether they represent each defined KA. If the ontology contained classes or properties related with the knowledge encompassed by the KA, the KA was considered as represented. The domain experts collaborated with ontology engineers to give additional explanations about the knowledge encompassed by KAs. It is worth mentioning that if a “child” KA was represented by the ontology, the “parent” KA that represents more abstract knowledge was also considered as represented.

Second, a CVA of existing ontologies was conducted to determine whether the KAs were common to Smart Grid scenarios. In particular, the *application-requirements matrix* technique proposed by Pohl et al. [9] was applied (taking as reference application-requirements matrix applied by Moon et al. [21]) to determine whether the KAs are common to Smart Grid scenarios depending on how many ontologies represent them. One application-requirements matrix was created to classify the KAs of each subdomain. As an example, Table 1 shows the application-requirements matrix of a set of KAs of the *energy consumption systems* and *device operation subdomains* (34 KAs were defined in total for these subdomains). The left column contains the KAs of the subdomain. The top rows list the Smart Grid scenarios and the energy ontologies classified by the Smart Grid scenarios they support (this classification can be consulted at [12]). To simplify the matrix, we have omitted a couple of ontologies. The matrix indicates if an ontology represents a KA (‘X’) or not (‘-’). With this information, we could deduce which Smart Grid scenarios reuse each KA. We considered that a Smart Grid scenario reuses a KA if the KA is represented by at least one ontology developed to support the Smart Grid scenario. KAs were classified into common and variant according to their *commonality ratio* (CV ratio) (right column in Table 1): the ratio of the number of Smart Grid scenarios that reuse the KA to the total number of Smart Grid scenarios. In particular, 75% was used as the threshold value of the CV ratio to classify the KAs. The KAs equal or above the threshold were considered as

TABLE 1: APPLICATION-REQUIREMENTS MATRIX

Ontologies Knowledge areas	Smart Grid scenarios								Commonality ratio
	Smart Home energy management			Building/district/city energy management		Organization energy management		Smart Grid Demand Response management	
	ThinkHome ontology	EnergyUse ontology	SAREF4EE ontology	SEMANCO ontology	BOOnSAI ontology	DEFRAM project ontology	DERI Linked dataspace	ProSGV3 ontology	
Appliances	X	X	X	X	-	X	-	X	100%
Brown goods	X	X	-	-	X	-	X	X	100%
White goods	X	X	X	X	-	-	-	X	75%
Refrigeration devices	X	X	-	-	-	-	-	X	50%
Device energy consumption	X	X	X	-	X	-	X	X	100%
Energy consumption systems operation	X	X	-	-	-	-	-	-	25%
Appliance working mode	-	-	X	-	-	-	-	-	25%

TABLE 2: CVA AT APPLICATION TYPE LEVEL

	Smart Home energy management		
	Home energy assessment	Home energy saving advice	Home appliances Demand Response management
Ontologies	ThinkHome ontology	EnergyUse ontology	SAREF4EE ontology
Knowledge areas			
Energy consumptions systems operation	X	X	-
Appliance working mode	-	-	X

common, while the rest were considered as variant.

Third, each KA was classified into one layer according to the CVA results. The common KAs were placed in the *common-domain layer*. Variant KAs reused in more than one Smart Grid scenario were assigned to the *variant-domain layer*. The KAs reused only in one Smart Grid scenario were assigned to one of the sublayers of the *domain-task layer* according to a CVA at the application type level. The KAs reused by more than one application type of a Smart Grid scenario are likely to be reused in more application types of that scenario and were placed in the *Smart Grid scenario sublayer*. The KAs reused only by one application type were assigned to the *application type sublayer*. Following the sample CVA of Table 1, the *energy consumption systems operation* and the *appliance working mode* KAs were reused only by Smart Home energy management applications. Thus, they were included in the CVA at application type level (Table 2). The *energy consumption systems operation* KA was reused by more than one Smart Home energy management application type. Hence, it was placed in the *Smart Grid scenario sublayer*. The *appliance working mode* KA was reused only by one Smart Home energy management applications and placed in the *application type sublayer*.

D. Step 4: Definition of the Ontology Modular Structure

In this step, the ontology engineers structured the knowledge of each layer into ontology modules to complete the ontology design. This step was performed taking as reference the ontology modularization principles applied by the main reusable and usable ontology design methods: loosely coupling and self-containment [5]. One module was defined for each KA and placed in one ontology layer/sublayer according to the CVA results. The modules were related according to the knowledge dependencies defined in Step 2. The modules of the *Smart Grid scenario* and *application type* sublayers were classified into the Smart Grid scenario/application types where the KAs they represent are reused.

V. EVALUATION

The ontology design method presented in Section 4 was evaluated by domain experts and ontology engineers. A group of domain experts and ontology engineers conducted Steps 1 and 2 and different ontology engineers (eight in total) conducted Steps 3 and 4 to design parts of DABGEO. Each ontology engineer performed Steps 3 and 4 individually in a blind

process. A survey was performed to capture the knowledge classification obtained by each ontology engineer. The survey also included a questionnaire to identify the main advantages and improvement aspects of the proposed method. This questionnaire can be found online in the appendix: https://innoweb.mondragon.edu/innoweb/questionnaire_SPL_ontology_method.pdf.

Fig. 5 shows the number of modules defined by each engineer for each ontology layer. It also shows the number of modules of the *domain-task layer* that were classified into each energy management application type. It is worth mentioning that the designed ontology parts were limited to support three application types (shown in Fig. 5). In general terms, the number of modules defined by each ontology engineer was similar in all layers. This similarity is due to the high *degree of consensus* with which the ontology engineers classified the defined KAs into layers. We understand by degree of consensus of a KA the percentage of ontology engineers that classified the KA into the same layer. The average degree of consensus of the KAs was 76%. Therefore, there was a high consensus when classifying the common and variant domain knowledge. In addition, 81% of the KAs that were classified into one application type within the *domain-task layer* by most of ontology engineers, were not classified into other application types by other ontology engineers. Hence, the knowledge reused only by specific application types was identified.

Regarding the questionnaire, we received eight responses from participants of the proposed method evaluation. 100% respondents considered that the application of SPL engineering techniques was useful and 80% would recommend the proposed method to design reusable and usable ontologies in other domains apart from the Energy. According to the respondents, the main benefits of the proposed method are the following: (1) it provides clear and mechanical steps to classify the ontology domain knowledge taking as reference existing ontologies and (2) the CVA of existing ontologies provides a detailed classification of the knowledge reused by specific application types, while keeping separate the knowledge reused by most applications. On the other hand, the main improvement aspect deals with the required manual effort. Although the proposed method prevented from classifying the domain knowledge from scratch, it required a significant manual analysis effort to check whether each KA is represented by existing ontologies.

VI. LESSONS LEARNT

Considering the similar knowledge classifications obtained in Section 5, domain experts and ontology engineers could apply the steps of the proposed method to (1) perform a CVA of existing ontologies and (2) classify the domain knowledge into different layers. The evaluation participants did not need to perform an analysis of the requirements of each application type to classify the common and variant domain knowledge of DABGEO from scratch. In addition, they considered the proposed method useful and easy to follow. Hence, we can state that the CVA of existing ontologies complements domain experts and ontology engineers' expertise when designing

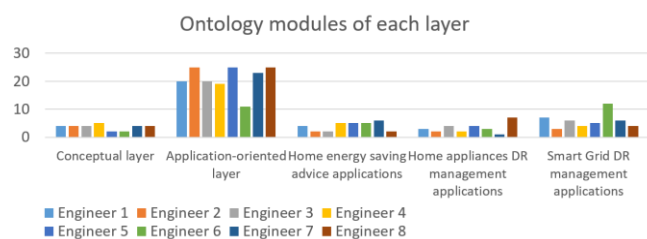


Figure 5: Ontology modules of each layer

reusable and usable ontologies in complex domains, thus saving ontology design effort. In addition, the ontology engineers identified the variant domain knowledge reused in specific applications. Thus, we can state that the CVA of existing ontologies enables an accurate domain knowledge classification. Bearing in mind these benefits, ontology developers should consider applying SPL engineering techniques when designing ontologies that (1) will be reused in different applications and (2) represent complex domains. In particular, existing ontologies should be identified and their knowledge should be divided and classified into different abstraction levels. Then, the ontologies should be analyzed to classify the domain knowledge based on their knowledge similarities and differences through a CVA.

Despite of these promising results, the CVA should be conducted with tool support to automate the process of checking whether certain KAs are represented by existing ontologies. In particular, tools that check (semi)automatically if a set of CQs are answered by ontologies should be developed. These tools can take as input the CQs encompassed each KA to check whether existing ontologies represent the KAs [18].

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have shown the experiences of applying SPL engineering and ontology design techniques in combination to design DABGEO, a reusable and usable global energy ontology. The proposed method analyses the knowledge similarities and differences of existing ontologies to classify the common and variant domain knowledge into different layers. The method was applied by domain experts and ontology engineers to design part of DABGEO. The results show that that the application of SPL engineering techniques enables a systematic and accurate domain knowledge classification that complements domain experts and ontology engineers' expertise. Hence, ontology design effort of reusable and usable ontologies is saved. Bearing in mind these benefits, the proposed approach should be applied to design ontologies that will be reused in different applications and represent complex domains.

Our current work is focused on defining a methodology based on the proposed method, so that it can be applied and replicated in other complex domains apart from the Energy. We are working on describing in detail and generalizing each step. The medium-term work will focus on incorporating tool support to reduce the manual ontology analysis effort.

REFERENCES

- [1] T. Gruber, *Ontology*. Springer, 2009.
- [2] H. Chen, F. Perich, T. Finin, and A. Joshi, "Soup: Standard ontology for ubiquitous and pervasive applications," in *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, 2004, pp. 258–267.
- [3] H. Wache, T. Voegelé, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner, "Ontology-based integration of information—a survey of existing approaches," in *IJCAI-01 workshop: ontologies and information sharing*, 2001, vol. 2001, pp. 108–117.
- [4] J. Morbach, A. Wiesner, and W. Marquardt, "OntoCAPE—A (re) usable ontology for computer-aided process engineering," *Computers & Chemical Engineering*, vol. 33, no. 10, pp. 1546–1556, 2009.
- [5] M. d Aquin, "Modularizing ontologies," in *Ontology Engineering in a Networked World*, Springer, 2012, pp. 213–233.
- [6] J. Morbach, A. Yang, and W. Marquardt, "OntoCAPE: A large-scale ontology for chemical process engineering," *Engineering applications of artificial intelligence*, vol. 20, no. 2, pp. 147–161, 2007.
- [7] D. Thakker, V. Dimitrova, L. Lau, R. Denaux, S. Karanasios, and F. Yang-Turner, "A priori ontology modularisation in ill-defined domains," in *Proceedings of the 7th International Conference on Semantic Systems*, 2011, pp. 167–170.
- [8] P. Spyns, Y. Tang, and R. Meersman, "An ontology engineering methodology for DOGMA," *Applied Ontology*, vol. 3, no. 1–2, pp. 13–39, 2008.
- [9] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.
- [10] A. Fantechi, S. Gnesi, I. John, G. Lami, and J. Dörr, "Elicitation of use cases for product lines," in *International Workshop on Software Product-Family Engineering*, 2003, pp. 152–167.
- [11] E. Curry, W. Derguech, S. Hasan, C. Kouroupetroglou, and U. ul Hassan, "A Real-time Linked Dataspaces for the Internet of Things: Enabling 'Pay-As-You-Go' Data Management in Smart Environments," *Future Generation Computer Systems*, vol. 90, pp. 405–422, 2019.
- [12] J. Cuenca, F. Larrinaga, L. Eciolaza, and E. Curry, "Towards Cognitive Cities in the Energy Domain," in *Designing Cognitive Cities*, Springer, 2019, pp. 155–183.
- [13] N. Guarino, "Semantic matching: Formal ontological distinctions for information organization, extraction, and integration," in *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology*, Springer, 1997, pp. 139–170.
- [14] A. Gomez-Perez, M. Fernández-López, and O. Corcho, *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer Science & Business Media, 2006.
- [15] K. Czarnecki, C. Hwan, P. Kim, and K. Kalleberg, "Feature models are views on ontologies," in *10th International Software Product Line Conference (SPLC '06)*, 2006, pp. 41–51.
- [16] T. Asikainen, T. Männistö, and T. Soinen, "Kumbang: A domain ontology for modelling variability in software product families," *Advanced Engineering Informatics*, vol. 21, no. 1, pp. 23–40, 2007.
- [17] C. Quinton, N. Haderer, R. Rouvoy, and L. Duchien, "Towards multi-cloud configurations using feature models and ontologies," in *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, 2013, pp. 21–26.
- [18] M. C. Suárez-Figueroa, "NeOn Methodology for building ontology networks: specification, scheduling and reuse," *Informatica*, 2010.
- [19] F. B. Ruy, G. Guizzardi, R. A. Falbo, C. C. Reginato, and V. A. Santos, "From reference ontologies to ontology patterns and back," *Data & Knowledge Engineering*, 2017.
- [20] A. Harhurin and J. Hartmann, "Service-oriented commonality analysis across existing systems," in *2008 12th International Software Product Line Conference*, 2008, pp. 255–264.
- [21] M. Moon, K. Yeom, and H. S. Chae, "An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line," *IEEE transactions on software engineering*, vol. 31, no. 7, pp. 551–569, 2005.

Identify MVC architectural pattern based on ontology

Qiang Yin, Lulu Wang, Bixin Li

School of Computer Science and Engineering
Southeast University

Nanjing, China

{220163720, wanglulu, bx.li}@seu.edu.cn

Abstract—MVC architectural pattern is widely used in software architecture design. It helps decouple the processing and the visualization of system data. Identified MVC architectural pattern helps understand how the software is actually implemented based on MVC architectural pattern, and further improve the consistency between design and source code. This paper proposes an ontology-based MVC architectural pattern identification method. Firstly, we use the combination of design patterns to describe the structure of MVC architectural pattern, so as to construct the MVC ontology of concept layer. Then we construct a program dependency graph by extracting the dependencies between entities in the target system, and build the ontology of instance layer. Finally, the MVC architectural pattern ontology of the specific target system is inferred by ontology reasoner in order to obtain MVC architectural pattern and the pattern elements included in each component. We use open source projects as the benchmark, and the experimental results show that our method effectively identifies the MVC architectural pattern and the pattern elements in software system.

Keywords—MVC; Architectural Pattern; Pattern identification; Observer Pattern; Strategy pattern; Ontology

I. INTRODUCTION

Architectural style is often used to describe the architecture [1], the architectural style is also known as the software architectural pattern. MVC is a software architectural pattern that widely used in desktop applications and Web information systems [3], which greatly improved the speed and stability of the system, also makes the software easy to maintain and easy expansion. But with the evolution of software, software design changes and the loss of software documentation make it difficult for developers to understand the architectural pattern of software. Such software is often difficult to maintain. Understanding the architectural pattern of the system is very helpful for the maintenance work. Bass et al. mention that some 80% of all costs in software development are related to maintenance activities [2]. Therefore, it is very meaningful to research the MVC software architectural pattern identification.

This paper studies the identification of MVC architectural pattern. At present, there is no set of theory and technology at home and abroad to support MVC architectural pattern identification. But there are some studies on the identification of architectural pattern. J. Paakki et al. regarded the problem that mining pattern from UML diagrams as a constraint satisfaction problem in 2000 [4]. J. Peters proposed an architectural pattern matching method based on Semantically Rich Modular Architecture [5], using genetic algorithm to match

pattern instance. M. Lungu proposes an architectural pattern identification method that uses a source code structure as a pattern and builds on a lower-level pattern extracted from source code, iteratively and interactively produces a more advanced view [6]. H. Yan introduced the tool DiscoTect, which identifies the architecture pattern of the runtime object-oriented system [7]. Mavridou Anastasia pointed out that architecture can be represented by logic, and architectural style can be described by configurations [8]. Thomas Haitzer proposed a semi-automated architectural pattern identification method based on architecture primitives [9]. The Service-oriented Architecture pattern identification method based on model checking was proposed by Penta and Sandonep in 2007 [10].

From the above related works, it can be concluded that the identification of architectural pattern requires two aspects of work: describing the architectural pattern and matching the description of the architectural pattern with the source code. However, the related work still has the following shortcomings.

- Lack of automation, the above methods are not highly automated, which makes it difficult to implement.
- Existing research does not identify specific pattern elements in the software system and does not provide a more detailed reference for developers and maintenance personnel. Pattern element is the pattern-related class in software system.

This paper proposes an ontology-based MVC architectural pattern identification method, which can automatically identify the MVC pattern instances in a specific target system, the pattern instances include the MVC architectural pattern and the pattern elements. The contributions of this paper mainly include the following three points. Firstly, the observer design pattern and the strategy design pattern are used to represent the MVC architectural pattern composition principle, and the ontology formally describes the MVC architectural pattern, the observer pattern and the strategy pattern. Secondly, the ontology inference engine is used to match the formal description of the MVC architectural pattern with the specific target system. The ontology inference engine can automate this process. Finally, read the source code and documentation of target system manually is used to verify the correctness of the identification result and effectiveness of the method. The recall rate, the precision and the F1-measure of the MVC architectural pattern and pattern elements are calculated to measure the effectiveness of the method.

This paper is structured as follows: In the section 2, the MVC architectural pattern and ontology are introduced. In the

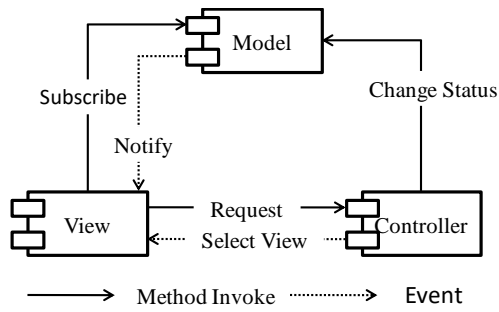


Fig. 1. MVC architectural pattern ^[17]

section 3, the research method of this paper is introduced. In the section 4, the experiment and evolution is introduced. Section 5 contains conclusion and future work.

II. BACKGROUND

A. MVC architectural pattern

The Model-View-Controller(MVC) architectural pattern divides an interactive application into three components ^[11]. First created by Trygve R and implemented in the Smalltalk-80 environment. The model contains core functions and data; the view displays information to user; the controller processes user's input. Both the view and controller form the user interface, and the change propagation mechanism ensures consistency between the user interface and the model.

MVC decouples views and models by establishing a subscribe/notify protocol between them ^[12]. A view must ensure that its appearance reflects the state of the model. Whenever the model's data changes, the model notifies views that depend on it. In response, each view gets an opportunity to update itself. So, the model-view relationship is an example of the Observer design pattern. The View-Controller relationship is an example of the Strategy design pattern. Encapsulate the response mechanism in the Controller object. To implement different response strategies, simply replace them with different kinds of Controller instances. Fig.1 shows MVC architectural pattern.

B. Ontology

In computer science, ontology refers to "a formal, clear and detailed description of a shared conceptual system" ^[13]. Ontology is used to express related concepts, entities and formal relationships between specific domains. The ontology can effectively describe the abstract relationships between specific domain concepts, and ontology can also reason in order to excavate the implicit relationship between abstract concepts. Ontologies generally use ontology language for related expression work. Currently, description logic (DL) and framework logic (FL) ^[15] are generally used to define ontology. In this paper, since the description logic is used to define the ontology, the DL is introduced below.

DL is a formal language used for the description of ontology, the relationship between the concepts of ontology and the relationship between individuals of concepts. Description logic has excellent expressive ability and logic-based inference ability, and the description logic has clear

inference algorithms ^[14]. The W3C international standard OWL and RDF language is based on the DL. The description logic system contains three parts: concept, role and individual. The assertion between concepts is called TBOX, and the assertion between individuals is called ABOX.

1) TBOX

TBOX is a set of axioms in description logic, which contains the connotation knowledge of the application domain, and uses the terminological axioms to describe the assertions between concepts. The general term axiom has two forms: inclusion and equality.

2) ABOX

ABOX is a set of description logic used to describe the relationship between individuals. It is an extension of the concept in the application domain to instantiate assertions axiom of individual and facts between individuals. Generally, there are two forms of instantiation axioms: concept assertion and role assertion.

The ontology of concept layer in this paper is to express through the OWL language, and the ontology of instance layer is to express through the RDF language, in order to build the ontology of MVC architectural pattern for a specific target system.

III. METHODOLOGY

The ontology-based MVC architectural pattern identification method proposed in this paper aims to identify the MVC architectural pattern and pattern elements in open source software. The method is mainly for the project that developed by JAVA. The method is mainly divided into three steps. Fig. 2 shows the overall process of the method.

- The first step is to represent the MVC architectural pattern as the observer pattern and the strategy pattern, and then describe the MVC architectural pattern formally by using the DL in order to form the MVC architectural pattern ontology of conceptual layer, the ontology of conceptual layer is also the Tbox in DL that mentioned in the second chapter.
- The second step is to use the code parsing tool to extract the dependencies between the entities in the source code, and use the RDF language to represent the dependencies of the entities in the source code into triples to form ontology of instance layer, the ontology of instance layer is also the Abox in DL that mentioned in the second chapter. Then, combining the MVC architectural pattern ontology of concept layer with the ontology of instance layer to form the MVC architectural pattern ontology of specific target system;
- The third step is to derive the expected result from the MVC architectural pattern ontology of the specific target system using the ontology inference engine, including whether the MVC architectural pattern is applied and the MVC architectural pattern elements in the target system.

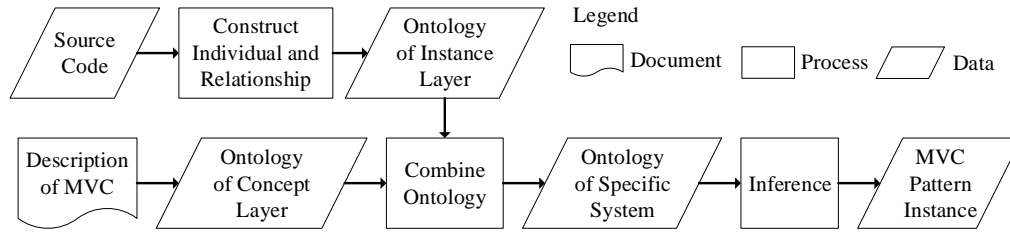


Fig. 2. The process of our method

A. Ontology of concept layer

The MVC pattern consists of three components: model, view, and controller. The pattern separates the model and view through the Observer pattern. The view is the observer in the Observer pattern. The model is the subject in the Observer pattern. Once the model data changes, the model will inform the relevant view. MVC controller is a component that accepts user input and responds to user input. The Strategy pattern is implemented, the view is the Context in the Strategy pattern, and the controller is the strategy in the Strategy pattern. On the other hand, most Java GUI applications use the Swing toolkit or the AWT toolkit, so the classes in View must inherit the base classes in JavaSwing and AWT. According to this feature, you can identify the specific class of the view in MVC.

The identification of the MVC pattern is transformed into the identification of the design pattern. The ontology of constructing the MVC architectural pattern is to construct a combination of two design pattern ontology and transform the abstract architectural pattern into a specific design pattern, thereby mapping the relationship between the pattern components to specific source code. Table I shows the DL of the main part of the ontology of concept layer.

B. Ontology of specific target system

Building the MVC architectural pattern ontology of a specific target system is mainly divided into two steps. Step one is to extract information for the target system. The purpose of information extraction is to extract the dependencies between entities in the target system in order to construct Program Dependence Graph (PDG). The entity includes methods and classes; in step two, the PDG is converted into an RDF triples, and the individual in the DL is added to the MVC architectural pattern ontology of concept layer to form an MVC architectural pattern ontology of the specific target system.

1) extract information

This paper is mainly for the Java open source project, using JAVA Development Tool(JDT) to complete the information extraction work. The main flow of JDT extraction is to first generate the Abstract Syntax Tree (AST) of the source code, and then construct the PDG according to the information on the AST.

The construction of program dependency graphs is to obtain the dependencies between classes and methods by traversing the AST. Constructing a PDG for the method of this paper requires extracting the following information, as shown in Table II.

TABLE I. DL OF THE ONTOLOGY

DL of MVC pattern: $MVCPattern \equiv \exists \text{containsElement.Model}$ $\cap \exists \text{containsElement.View}$ $\cap \exists \text{containsElement.Controller}$
DL of Observer pattern: $ObserverPattern \equiv \exists \text{containsElement.AbstractSubject}$ $\cap \exists \text{containsElement.ConcreteSubject}$ $\cap \exists \text{containsElement.AbstractObserver}$ $\cap \exists \text{containsElement.ConcreteObserver}$ $AbstractSubject \equiv Interface$ $\cap \exists \text{containsElement.AbstractNotify}$ $ConcreteSubject \equiv NomalClass$ $\cap \exists \text{containsElement.ConcreteNotify}$ $AbstractObserver \equiv Interface$ $\cap \exists \text{containsElement.AbstractUpdate}$ $ConcreteObserver \equiv NomalClass$ $\cap \exists \text{containsElement.ConcreteUpdate}$
DL of Strategy pattern: $StrategyPattern \equiv \exists \text{containsElement.AbstractStrategy}$ $\cap \text{containsElement.ConcreteStrategy}$ $\cap \text{containsElement.Context}$ $AbstractStrategy \equiv Interface$ $\cap \exists \text{containsElement.AbstractAlgorithm}$ $ConcreteStrategy \equiv NomalClass$ $\cap \exists \text{containsElement.ConcreteAlgorithm}$

TABLE II. TYPE OF DEPENDENCY

Dependency Edge	Master	Slave
extend	Class	Class
implement	Class	Class
composite	Class	Class
invoke	Method	Method
instantiation	Method	Class
aggregation	Class	Class

2) Building ontology

Firstly, convert PDG to RDF triples, RDF (Resource Description Framework) uses the triples to represent the relationship between entities, which is divided into three parts: subject, predicate and object. It effectively describes the relationship between classes, methods and other entities in the source code, the RDF triples can be regarded as ABOX in the DL. The subject indicates that the vertex of master in the graph, the predicate indicates the dependency in the PDG, and the object indicates the vertex of slave in the PDG; then, add the RDF triples to the ontology of the MVC architectural pattern concept layer, where the subject and object of the RDF triples are regarded as the individual in the DL, the predicate of the RDF triple corresponds to the role in DL. After adding RDF triples, the MVC architectural pattern ontology of the specific target system is formed.

C. Inference the MVC architectural pattern

Inference is essentially the process of drawing conclusions from existing facts according to certain rules^[16]. The inference process is mainly applied to the ontology inference machine. The inference engine mainly has two functions: ontology conflict detection and acquisition of implicit knowledge. Ontology conflict detection is to ensure the logical consistency between classes and individuals in the ontology. Obtaining implicit knowledge is that inference unknown knowledge according the established rule. The current inference engines are mainly divided into two categories, one is the inference engine based on the traditional description logic. The representatives of such inference engines are RACER, FaCT++, Pellet, et al. Others the inference engine are based on inference rules. Representatives are Jena, Jess. The inference engine mainly used in this paper is the Jena inference engine.

Inferring the MVC architectural pattern ontology of specific target system is to infer the existence of the pattern instances in the ontology through the inference rule. The key step of inference is to write the reason rules. The inference rules are written according to the MVC architectural pattern composition principle. The main process is to derive the unknown facts defined in the MVC concept layer ontology based on known facts such as individual and object attributes. Individuals are classified into classes defined by the concept layer of MVC architectural pattern.

According to the MVC architectural pattern concept layer ontology, we can divide the inference rules into five parts. The first part of the inference rules is written by JavaSwing inheritance rules, the purpose is to identify the classes related to GUI. The second part of the inference rules use the composition principle of observer pattern in order to identify View and Model. The third part of the inference rule is written by the principle of strategy pattern, which is to identify View and Controller. The fourth part of the inference rule is based on the inference results of the previous three steps to infer the MVC architectural pattern and the pattern elements. The fifth part of the inference rule is common rules. As shown in Table III, it is the inference rule of the observer pattern.

TABLE III. RULES OF OBSERVER PATTERN

Rules of Observer pattern: [observer_rule:(?observer rdf:type JAVA:NormalClass), (?observer JAVA:isA ?Aobserver), (?Aobserver JAVA:contain ?Aupdate), (?observer JAVA:contain ?update), (?update JAVA:override ?Aupdate), (?subject JAVA:isA ?Asubject), (?subject rdf:type JAVA:NormalClass), (?subject JAVA:aggregation ?Aobserver), (?Asubject JAVA:contain ?Anotify), (?subject JAVA:contain ?notify), (?notify JAVA:override ?Anotify), (?notify JAVA:invoke ?Aupdate) -> (?observer rdf:type OB:ConcreteObserver), (?Aobserver rdf:type OB:AbstractObserver), (?subject rdf:type OB:ConcreteSubject), (?Asubject rdf:type OB:AbstractSubject)]
--

IV. EXPERIMENT AND EVALUATION

A. Purpose and metrics

The purpose of experiment is to verify the effectiveness of our pattern identification method, we evaluate the effectiveness of our research method through three metrics: Precision, Recall, and F1-measure. Finally, analyzing the advantages and disadvantages of the method through experimental data. The calculation formula for the three metrics is as follows:

- Precision=(TP)/(TP + FP)
- Recall= TP / (TP+ FN)
- F1-measure=2*Precision*Recall / (Precision + Recall)

B. Experimental setup

The method of this paper is implemented in Java language. The operating system is macOS 10.12.6, the JDK version is 1.8.0_121, and the development platform is Eclipse neon.2. The MVC architectural pattern ontology is constructed by Protégé, the information extraction tool is JDT, the specific target system MVC architectural pattern ontology is constructed by Jena, and the ontology inference is completed by Jena inference engine.

C. Result of Overall

In this section, we identify 20 popular open source projects, and then verify the correctness of the identification results and the effectiveness of the pattern identification method by analyzing the official documents and source code of 20 projects. Determine whether the project adopts the MVC architectural pattern by reading source code and official documentation. Finally, experimental data is obtained by comparing the recognition results with the verification results.

The results of the identification of 20 excellent open source projects are shown in the table IV. The first column of the table indicates the project name, the second column indicates the identification result, √ indicates that the identification result is MVC, × indicates that the identification result is not MVC, and the third column indicates that actual pattern of the project, √ indicates that the project adopts the MVC architectural pattern, and × indicates that the project does not adopt the MVC architectural pattern.

Through the results in the table V, the statistical data is shown in table V.

According to the calculation formula of Precision, Recall, and F1-measure, the following results can be calculated:

- Precision=(TP)/(TP + FP) = 0.75
- Recall= TP / (TP+ FN) =1.00
- F1-measure=2*Precision*Recall / (Precision + Recall) =0.86

According to the above results, the Precision is 0.75 and the Recall is 1.00, which indicates that the method in this paper can effectively identify whether the MVC architectural

TABLE IV. IDENTIFY RESULTS OF TWENTY PROJECTS

Project Name	Identify Result	Validation Result
clone	✓	✓
filezilla	×	×
freecol	✓	✓
terrier	✓	✓
mockito	×	×
lionengine	×	×
symphony	×	×
vert.x	×	×
okhttp	✓	×
mybatis	×	×
jnativehook	×	×
jadx	✓	×
jfinal-master	×	×
overlap2d	×	×
StormPlane	✓	✓
jenkins-master	×	×
shiro	×	×
pinpoint	×	×
latexdraw	✓	✓
MARIO	✓	✓

TABLE V. DATA OF IDENTIFY RESULTS

TP	FP	TN	FN
6	2	11	0

pattern is used in the software system, but there are still false positives.

D. Result of Part

In this section, we present three open source projects that use the MVC architectural pattern to verify the correctness of the pattern elements and the effectiveness of the method. The three open source projects are Clone, Mario, and Terrier. Firstly, it introduces the overall situation of three project. Then it gives the identification result of the three project. The identification result is the class related to the MVC pattern in the project, which is the pattern element. Then the paper understands the three projects by manually reading the source code, and classifies the classes related to MVC pattern in three projects. Finally, experimental data is obtained by comparing the identification results with the verification results.

Clone is a strategic game based on the MVC architectural pattern, which developed by Java. Mario is a popular adventure game developed in Java, similar to the Super Mario developed by Nintendo. Terrier is a highly flexible, efficient, and effective open source search engine, readily deployable on large-scale collections of documents. The basic information of the source code of three projects is shown in Table VI.

For the above three projects, due to the limitations of the paper length, we only give the identification results of Clone. The identification results of Clone are shown in Table VII.

In Table VII, classes in bold indicate false positives. The statistics of the identification results of the three projects are shown in Table VIII.

E. Analysis and Conclusions

By counting the identification results of 20 projects, we calculated that Precision is 0.75, Recall is 1.00, F1-measure is 0.86. The F1-measure is relatively high. Therefore, we can draw the conclusion that the method in this paper can effectively identify whether the MVC architectural pattern is

TABLE VI. BASIC INFORMATION ABOUT THREE PROJECTS

Project Name	Language	Files	Code Line
Clone	Java	91	10830
Mario	Java	282	32859
Terrier	Java	561	49823

TABLE VII. IDENTIFY RESULTS OF CLONE

Component	Pattern Element
Model	model.BoardModel; boardobject.Firildak; physics.GeometryReference ; boardobject.Ball; physics.GeometryInterface; physics.SimpleGeometry ;model.Player; boardobject.Cezerye; boardobject.LeftTokat; boardobject.RightTokat; boardobject.Gizmo; physics.GeometryCompare; boardobject.Cezmi; boardobject.Observable; physics.GeometryImpl; boardobject.Takoz; boardobject.TriangleTakoz; boardobject.Tokat; boardobject.SquareTakoz;
View	physics.GeometryImpl ; observer.CezmiObserver; gui.ClickHandler; gui.BeginningPanel; gui.GameWindow; physics.GeometryInterface ; observer.GizmoObserver; observer.PlayerObserver; gui.BoardPanel; gui.BuildingPanel; physics.SimpleGeometry ;observer.BallObserver; observer.CezeryeObserver; physics.GeometryReference ; observer.WinPanelObserver; physics.GeometryCompare ;
Controller	controller.DeleteGizmoController; controller.AddGizmoController; controller.RotateGizmoController; controller.QuitController; controller.SaveController; controller.PauseController; controller.MoveGizmoController; controller.ResumeController; controller.LoadController; controller.Controller;controller.PlayController;

TABLE VIII. DATA OF IDENTIFY RESULTS

Project Name	TP	FP	FN	Precision	Recall	F1-measure
Clone	39	7	0	0.85	1.00	0.92
Mario	128	54	21	0.70	0.86	0.77
Terrier	412	123	82	0.77	0.83	0.8

used in the software system. But there are still false positives, the reasons for false positives are as follows:

- There is a false positive because the method uses a rigid rule to infer whether there is an MVC architectural pattern in the software system. It does not rule out that some software systems have dependencies that satisfy the inference rules, but not the MVC architectural pattern.

By counting the identification results of three projects, we can see that the F-measure of each project is higher than 0.7, and the value of recall is higher than precision. Therefore, we can draw the conclusion that the method of this paper can effectively identify the class related to the pattern in the software, which we call the pattern element. The main reason for the false positives is that some classes that satisfy the inference rules but are not related to the pattern are also identified. The main reason for the false negative is that some classes have the characteristics of the pattern elements, but these features are not implemented according to the standard of MVC architectural pattern.

F. Threats to validity

There are potential threats to validity of our results. The first threat to validity is the method has only been validated for projects developed in the Java language, and has not been validated for projects developed using other object-oriented languages.

The second threat to validity is the method only validates 20 popular open source projects, and the code size of these 20 projects is less than 500k. Sample data is not very comprehensive.

V. CONCLUSION AND FUTURE WORK

This paper proposes an ontology-based MVC architectural pattern identification method, which can automatically identify the architectural patterns and pattern elements of MVC in software. By combining the MVC architectural pattern ontology of concept layer and the ontology of instance layer, the MVC architectural pattern ontology of the specific target system is formed, and then the ontology is inferred to obtain the MVC architectural pattern and the pattern elements. Through the statistics and analysis of experimental data, we analyzed the causes of false negatives and false positives, and summarized the advantages and disadvantages of the methods. The novelty of this method is as follows:

- Applying the design pattern to the identification of the architectural pattern makes the description of the MVC architectural pattern closer to the source code and easier to understand.
- The method also identifies the pattern elements in the software and provides developers and maintenance personnel with a more detailed reference.
- The automatic identification of the MVC architectural pattern is realized, because the method can reuse the same set of MVC architectural pattern ontology of concept layer for different systems, needn't to generate a different set of descriptions for each specific target system, and the ontology inference engine can automatically complete the architectural pattern matching process.

In the future work, firstly, we must improve the identification precision and recall of the MVC architectural pattern and pattern elements. The components and pattern elements of the MVC architectural pattern are closely related to the functionality of the software. We should consider the functional information of the class in the process of identification, so that the process is no longer limited to rigid rules; in addition, the identification of variants of the MVC architectural pattern is also worth doing, such as MVP(Model-View-Presenter) architectural pattern, which are widely used in Android, MVVM(Model-View-ViewModel) pattern using WPF technology.

ACKNOWLEDGEMENT

This work is supported in part by the National Key R&D Program of China under Grant 2018YFB1003902, in part by the Cooperation Project with Huawei Technologies Co., Ltd., under Grant YBN2016020009, and in part by National Natural Science Foundation of China under Grant 61872078, Grant 61572126, Grant 61402103, and Grant 61572008. Special thanks to Dr. Renhao Xiong in ISEU and anonymous reviewers.

REFERENCES

- [1] G. Abowd, R. Allen, D. Garlan. Using Style to Understand Descriptions of Software Architecture[C]. ACM SIGSOFT symposium on Foundations of software engineering. ACM, 1993: 9-20.
- [2] L. Bass, P. Clements, R. Kazman. Software Architecture in Practice[M]. Boston, Massachusetts, USA: Addison-Wesley, 2012.
- [3] CAO Shuang, Su-ling Jia. Apply MVC Architecture Pattern to C/S System[J]. Computer Knowledge and Technology, 2007-10: 946-959.
- [4] J. Paakki, A. Karhinen, J. Gustafsson, et al. Software metrics by architectural pattern mining[C]. Proceedings of the International Conference on Software: Theory and Practice. Beijing, China: Kluwer, 2000: 325-332.
- [5] J. Peter, v. d. Werf. A genetic approach to architectural pattern discovery[C]. Proceedings of the European Conference on Software Architecture. Copenhagen, Denmark: ACM, 2016: 17.
- [6] M. Lungu, M. Lanza, T. Girba. Package patterns for visual architecture recovery[C]. Conference on Software Maintenance & Reengineering, Los Alamitos CA: IEEE, 2006: 185-196.
- [7] H. Yan, D. Garlan, B. Schmerl, et al. DiscoText: a system for discovering architectures from running systems[C]. In: Proceedings of the 26th International Conference on Software Engineering. IEEE Computer Society. Washington, DC, USA: IEEE, 2004: 470-479.
- [8] A. Mavridou, E. Baranov, S. Blidze, et al. Configuration logics: Modeling architecture styles[J]. Journal of Logical and Algebraic Methods in Programming, 2017, 86(1): 2-29.
- [9] T. Haitzer, U. Zdun. Semi-automatic architectural pattern identification and documentation using architectural primitives[J]. The Journal of Systems and Software, 2015, 102.: 35-57.
- [10] M. D. Penta, A. Santone, M. L. Villani. Discovery of SOA Patterns via Model Checking[C]. In: 2nd International Workshop on Service Oriented Software Engineering: In Conjunction with the 6th ESEC/FSE Joint Meeting. New York, USA: ACM, 2007. 8-14.
- [11] F. Buschmann, R. Meunier, H. Rohnert, et al. Pattern-Oriented Software Architecture Volume 1: A System of Patterns[M]. USA: Wiley, 1996.
- [12] E. Gamma, R. Helm, R. Johnson, et al. Design Patterns: elements of reusable object-oriented software[M]. USA: Wiley, 1995.
- [13] T. R. Gruber. A translation approach to portable ontology specifications[J]. Knowledge Acquisition - Special issue: Current issues in knowledge modeling, 1993, 5(2): 199-220.
- [14] F. Baader, D. Calvanese, D. McGuinness, et al. The Description Logic Handbook: Theory, Implementation and Applications[M]. UK, Cambridge: Cambridge University Press, 2003.
- [15] L. Farinas, A. Herzig. Interference logic = conditional logic + frame axiom[J]. International Journal of Intelligent Systems, 1994, 9(1): 119-130.
- [16] C. Pan, H. Gu. Ontology Reasoner and Its Application[J]. Computer Systems & Applications, 2010-09.
- [17] Z. F. Ren, H. Zhang, M. S. YAN, et al. Overview of the research in model-view-controller pattern[J]. Application Research of Computers, 2004, 10: 1-4.

Grouping Semantically Related Change-Sets to Enhance Identification of Logical Coupling

Neeraj Mathur, Sai Anirudh Karre, Y. Raghu Reddy

Software Engineering Research Center

IIIT Hyderabad, Telangana, India

neeraj.mathur, saianirudh.karre{@research.iiit.ac.in}, raghu.reddy@iiit.ac.in

Abstract—Identifying dependency between various artifacts in a large scale software system is a non-trivial task. As the software evolves, multiple artifacts like files, docs, classes, database scripts, etc., are likely to undergo change concurrently. Such artifacts tend to have a dependency between them, otherwise referred to as logical coupling. Researchers have used *Support and Confidence* as an association rule based measurement to predict the levels of logical coupling among the software artifacts. However, employing a single change on a software artifact can span across various closely related changes when many code contributors are working on the same change. Thus it is important to pre-process and group these semantically related change-sets before identifying logical coupling. In this paper, we propose a method to identify logical coupling and group semantically related change sets. We evaluate our method on real-world git repositories and document our observations.

Index Terms—Cosine similarity, dependency, logical coupling, repository mining, software evaluation, software maintenance, reverse engineering

I. INTRODUCTION

Large Software systems invariably comprise of artifacts written in different programming languages integrated by diverse technologies using a variety of implementation methods. Identifying the dependency between artifacts written in two different programming languages is a non-trivial task. For example, a JavaScript method may depend on a web service written in C#. Tracking such dependencies gets increasingly difficult over a period of time and as they tend to become the origins of defects in an overall software project.

Heuristically, identifying dependencies from code-revision history is considered to be lightweight than conducting a structural analysis of the entire artifact. In the case of code-revision history, a small amount of information is required to be analyzed in order to understand the dependency. Such information is typically stored via the log files of version control system like GIT, SubVersion, TFS etc. Almost in all cases, such dependencies are primarily documented in the form of a free-text comment. Thus, dependency analysis can be performed between two or more artifacts written in different languages without having trouble in parsing and analyzing the content of the artifacts. Logical coupling is one such implicit dependency observed between two or more software artifacts. It has been found that artifacts that are considered to be logically coupled artifacts when they change together

frequently during the evolution of a system [9]. It can reveal dependencies that are *not structural* and therefore are not present in the code or in the documentation.

The reliability of all the existing studies on logical dependencies is inherently connected to the accuracy of the approach used to identify such dependencies [14]. Version Control Systems (VCS), that are atomic-featured in their nature have a *change-set* - which is comprised of mutually checked-in files that result in a single commit. In general, software practitioners often rely on the existence of the atomic commit feature and consider the change-set as the actual set of files that were changed together by a code-developer while working on a given code-based task. In the case of multiple developers implementing the same change, such code-change can span across a series of consecutively connected and closely related individual change-sets. Therefore, by simply inspecting the change-sets in isolation may lead to incomplete or incorrect results with respect to the association of logical coupling.

In the past, researchers have proposed approaches to understand logical coupling. Grouping commits with the same authors using sliding time window concept was widely used [14]. However, semantic relationships between the artifacts - like the same work item or issue number, cosine similarity of two revision comments, etc. have not been explored by researchers. *Cosine similarity* is a popular measure in Information retrieval and Data Mining areas. It can be used to measure the similarity between two documents with respect to their textual content [11]. In this paper, we present our preliminary work by utilizing such techniques to identify the logical dependency between artifacts leading to an increase in change-set identification and accuracy. In this paper, we:

- suggest an approach to improve logical coupling detection using semantic relations drawn from the revision comments and cosine similarity.
- perform a preliminary evaluation of the proposed approach for grouping semantically related change-sets.

The rest of the paper is organized as follows. In Section 2, we introduce the cosine similarity and grouping semantically related change-sets. In Section 3, we present the results of our preliminary evaluation. In Section 4, we present some related work. Finally, in Section 5, we state our conclusions and future work.

TABLE I
TERM FREQUENCIES WEIGHT(S)

Term	SaS	PaP	WH
1. Term Frequencies Count			
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38
2. Log Frequency Weight			
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58
3. Weight After Length Normalization			
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

II. GROUPING CHANGE-SETS

Software Practitioners tend to work on code usually spread across multiple change-sets, with multiple developers working on it at the same time. To group these change-sets, the semantic relations of the change-sets' comments can be utilized. For large scale software projects, the following can be observed as grouping criteria for change-sets (i) if and only if two comments have higher semantic similarity (or) (ii) if and only if two comments have same referencing work item or an issue number mentioned in it.

A. Cosine similarity

Cosine similarity is considered as a common measure for similarity computation [6]. Cosine similarity is used to fetch the similarity of words with respect to input query in regards to the text documents queried. To perform this computation, both the input query and the documents are converted into their respective unit vector of words (\vec{x}_i, \vec{y}_i) . We use the below equation 1 to compute the cosine similarity here, where x_i, y_i are the term frequency (tf) weight of a unit vector (term) in the revision comment x, y. Term frequency is the number of times a term occurs in a revision comment.

$$sim(x, y) = \cos(\vec{x}, \vec{y}) = \vec{x} \cdot \vec{y} = \sum_{i=1}^{|V|} x_i y_i \quad (1)$$

Let's look at the case where we wish to calculate cosine similarity of the books: *Sense and Sensibility* (SaS), *Pride and Prejudice* (PaP) and *Wuthering Heights* (WH). Table I lists the Term Frequencies in each of the documents, Log Frequency Weight (calculated by formula " $w = 1 + \log_{10}(tf)$ ") and Length Normalized Weight. A vector can be length-normalized by dividing each of its components by its length. We use Level-2 normalization (commonly referred as Euclidean norm) as defined in equation 2:

$$\|\vec{x}\| = \sqrt{\sum_i x_i^2} \quad (2)$$

Finally we compute cosine similarity of documents as listed below:

- $\cos(SaS, PaP) \approx 0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0 \approx 0.94$
- $\cos(SaS, WH) \approx 0.79$

The resultant values can be used to assess the extent of similarity based on a predefined threshold values. In equation 3, we used '0.8' as a threshold to measure semantic similarity. We have conducted a preliminary data analysis using the available datasets and have considered 80% as a reasonable threshold for desirable similarity values. This value can be provided by the developers assessing the similarity. However, reducing the threshold can result in poor precision, where as increasing the threshold may result in poor recall. So, it is required for developers to follow the standard threshold which is widely accepted to avoid poor precision and poor recall.

$$x \overset{GC}{\rightsquigarrow} y \stackrel{def}{=} \begin{cases} 1 & \text{if } sim(x, y) \geq 0.8 \text{ \& } datediff(x, y) < \alpha \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

However, in some cases relying on the cosine similarity index can result in false positives. For example, in some project like SignalR project [4] developers tend to put generic comments like "*addressed code review comments, fixed formatting, made changes as per code review feedback*" for ongoing activities. As part of our study, we exclude these kind of change-sets from grouping. Also, we compare the time difference of change-sets before grouping them i.e. if the time difference is more than a few months then we consider such changes as not related.

B. Hash tags

Another technique for grouping change-sets is based on the hashtags associated with the commits of co-changed artifacts. For instance, if we consider the instances of commits from Table II of two large scale open source projects like SignalR [4] and NopCommerce [2] hosted on GitHub and Codeplex, we see that in the example-1, commits '203cafc' and '5ad051a' have similar comments. In case of example-2, the developers tend to associate work item or issue number in the comment for future references, like in Git repository, the hash tags #2376 are associated with the comments.

We group change-sets having same hash tag values based on the mathematical formulae given in equation 4.

$$x \overset{GC}{\rightsquigarrow} y \stackrel{def}{=} \begin{cases} 1 & \text{if } HashTags(x) \cap HashTags(y) \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

C. Our Approach

The proposed approach is described in Algorithm 1. As shown in the algorithm, the initial steps required to build a postings list [6] for each revision so as to help in identifying the change-sets having same tokens. Once the posting list is built using Algorithm 2, the entire revision history is looped through (shown in line numbers 6-12) to get semantically related change-sets and group them in a

TABLE II
MOTIVATING EXAMPLES

Example 1
Commit: 203cafc DumpingDisconnect.Net Comment: Removing handling Disconnect message in the .NET Client as the server no longer sends Disconnect messages.
Commit: 5ad051a DumpingDisconnect.JS Comment: Removing handling Disconnect message in the .JS client as the server no longer sends Disconnect messages.
Example 2
Commit: 9c762c4 #2376 Reject failed invocation with a single JS object representing HubExce...
Commit: f9bfbe3 #2376 Tests verifying HubException details are sent to clients
Commit: 35cfccb #2376 Flow HubExceptions which to clients even with detailed errors disabled

list. The ‘getRelatedChangeSets’ subroutine returns semantically related change sets for the current iteration of the revision based equation (3, 4). The change-sets are then added to a dictionary of the list of semantically grouped change-sets.

Algorithm 1: Change set grouping algorithm

Result: Grouped Change Sets List

```

1 procedure group_changeset
2 revisionLists ← git.GetRevisionListIterator();
3 while rev in revisionLists do
4   | addToPostingList(rev.Id, rev.Comment);
5 end
6 Map(groupId, revisionList) changeSetGroups;
7 int groupId ← 1;
8 while revision rev in revisionLists do
9   | revList ← getRelatedChangeSets(rev.comment);
10  | changeSetGroup.Add(groupId, revList);
11  | groupId++;
12 end
13 end procedure

```

Algorithm 2 describes the subroutine used to create Postings List Index. The revision id and the associated comment are passed as input parameters to this routine. The parameters are further processed to remove all characters except ‘[0-9a-b]w’ using regular expression match thereby tokenizing the string with words and their occurrences in the comments. Each token with its revision id to the Postings List Index is added to the postings list. The sample posting list is listed in Table III.

TABLE III
SAMPLE POSTING LIST

Token	Postings List
commerce	doc1, 4; doc2, 5;
tokenize	doc1, 50; doc3, 23;

Algorithm 2: Postings list builder subroutine

Result: Generate Postings List

Input: RevId and Comment

```

1 procedure addToPostingsList
2 // get hashmap of terms with frequency count
3 tokensList ← getTokens(comment);
4 while token in tokensList do
5   | if postingsList.get(token.key) then
6     | | postingsList.Get(token.key).Add(token);
7   else
8     | | postingsList.Add(token.Key, token);
9   end
10 end
11 end procedure

```

Algorithm 3 describes the subroutine responsible for returning semantically related change-sets. It accepts revision ‘comment’ as a parameter. It converts the comments to tokens and queries the postings list dictionary for these tokens to fetch the posting list. Each posting list returned is reviewed against the criteria mentioned in the equation to return the related change-set list.

Algorithm 3: Fetch semantically related change-set

Result: Generate Postings List

Input: Comment

```

1 procedure getRelatedChangeSets
2 tokens ← getTokensList(comment);
3 postingsList ← getPostingsList(tokens);
4 List < revID, Comment > revisionList = null;
5 while post in postingsList do
6   | bool isRelevant ← false
7   | if hasSameHashTag(comment, post) then
8     | | isRelevant ← true;
9   end
10  | if cosine(comment, post.comment) > 0.8 then
11    | | isRelevant ← true;
12  end
13  | if isRelevant then
14    | | revisionList.Add(post.revID, post.Comment);
15  end
16 end
17 end procedure

```

III. PRELIMINARY EVALUATION

In this section, we provide details of our preliminary evaluation using our proposed approach to group change-sets in few C# programs. we used NGit [1] to traverse the Git repository change-sets. We used two open source projects (SignalR and NopCommerce) that had substantial revision history to evaluate our approach. SignalR is an ASP.Net library that provides real-time communication support to a web application and NopCommerce is an ASP.Net based e-commerce web system.

TABLE IV
MANUAL EVALUATION OF IDENTIFIED GROUPING

NopCommerce			SignalR	
#	Appropriate?	Commits & Notes	Appropriate?	Commits & Notes
1	Yes	8bacf936baf4, efc731eeecb6, fd4ab9f67cce Enhancement for store owner to search unpublished and published products	Yes	ae9d5f7d57db, 8a2245b17d41 Modification for Forever Frame JS client
2	Yes	e6ec8e0b83ee, 782d3b87a6e3, c87537559940, d2792ada31c8 : Changes for product search and user friendly product name	Yes	0611ce61abe9, 261bb48fbca8 Changed the logic of addQs question mark query string detection
3	Yes	d40a89b56610, d0c04fc618d4 Modification related to custom validation	No	7996107ffbea, 877bcd59c454 Different instances of 'Removed unused code'
4	Yes	c5fe44ff76b8, 74e4a8e6c154, 28fb664d5c5f Modifications related to Shipping Address	Yes	240f6c58a5c3, de40de96a6b0, ff0bc98c2d34 Fix to ensure connection with LongPolling client
5	Yes	5e559b08a9ea, 050ddf2133e2 Enhancement related to filter shipments and Orders by warehouse	Yes	0b71d56, a60d923, 9c762c4, f9bfbe3e, 35cfccb Modifications related to HubExceptions
6	Maybe	305d4c1268b9, 3bf134f2c758, 9cdd0b2699b3 First two commits are related to store mapping to setting and third is related to store mapping to categories	Yes	8dc620093097, dd39b641bc27, 94ff30dd5821 Updates to crank for automation, and more Stress metrics
7	Yes	7a62bbdc9b24, c55d3897bf68 Localization changes for hard coded string	Yes	4148ea70f62d, 43aac84329b7 Handling of security errors in websockets
8	Yes	f020e98231f5, 95263d99979f Enhancement for friendly name of Affiliates	MayBe	a596aeb43c55, 5b47f9ba24a8 Updated self host sample.
9	No	188f4243ba1d, 8330d952235e First is the Fluent library update and second is the Entity Framework library update	Yes	143aa036367b, 7be649699e1d Changed Web Socket implementation to not send an empty frame at the end
10	Yes	57b7c7fdc445, 410eae6cd1f8 Modification for manufacturer store mapping	Yes	f81f6c2, 1118b15, cdaaa33, 3 5b65d0 Ensure JS SSE & WS transports will attempt reconnecting multiple times
11	Yes	d2f341323abc, fbd49df6f2dd, f02244dc5fae Enhancement for shipping rate computation for 'FedEx', 'UPS' and 'by weight'	Yes	4cf7d2b, 730aa53, c6ffb08, 41317c9, 56e4002, 8dc3ac4 :Exception handling unresolved endpoints in ServiceBus scale-out
12	No	bcecd04eb644, 5cfc2977138, b559ca0aa2e4 Author forgot a file to checkin in previous commit, but comments were same in all the commits	Yes	5ad051a9822b, 203cafdc1cbe Handling of disconnect message in JS and .Net
13	Yes	c3782eef4eba, b80d4cdecfa1 Added "Order paid" message template sent to a customer	Yes	1162f9163ba8, 47c7084ec3a4, ccdade4488ef added checks for null and empty values in the send and group methods in hub and persistent connection
14	-	-	Yes	bddcb70, 7dfa876, 13f2ffe, 9ff238e, 61c0c5c, 212fb4e Modification as per the static code analysis tool 'FxCop', Usually these are not related changes as FxCop is used to check naming conventions

We calculated the basic descriptive statistics for the number of commits grouped together. As listed in Table V, in case of NopCommerce project, 357 change-set groups were created with 1017 commits which are 19.44% of the actual commits with a maximum of 11 revisions and an average of 2.85 commits per change-set. In SignalR project, 301 change-set groups were created with 969 commits which are 21.49% of the actual commits with a maximum of 22 revisions and an average of 3.21. The figure 1 displays a Scatter Plot view of length versus number of change-set groups created for NopCommerce and SignalR projects respectively. Overall, there is a little dispersion in the values as evident by the standard deviation.

To corroborate the results of our algorithm, we have made attempts to conduct an inspection of a few random samples [3]

TABLE V
CHANGESSET GROUP STATISTICS

Project	NopCommerce	SignalR
Total commits	5229	4509
Active Since	2009	2011
# of groups	357	301
# of commits grouped	1017	969
% of commits grouped	19.44	21.49
Max	11	22
Avg	2.85	3.21
StDev	1.48	2.7

from the 658 change-set groups identified by our algorithm. We inspected 13 samples from each NopCommerce and SignalR to verify whether the revisions were grouped to a single

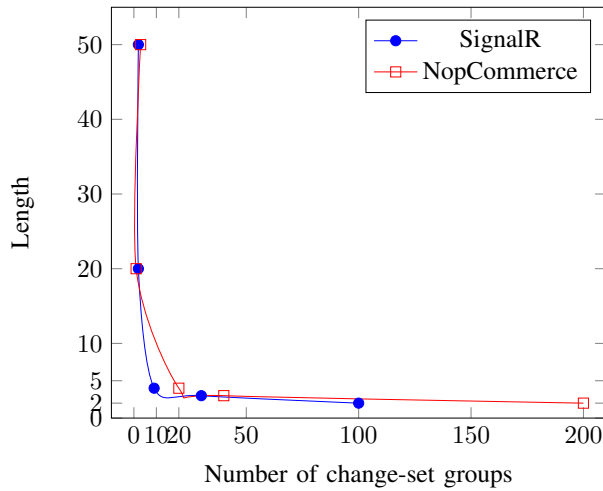


Fig. 1. Scatter plot of length versus change set groups

purpose or not. By verifying the revision files, comments, and diff code - we were able to judge whether the grouping made sense. We have used a statistical random sample size calculator [3], which enabled us to generalize our results with a margin of an error of 20% and confidence level of 80%. The results of our inspection are listed in Table IV.

In ‘NopCommerce’, out of 13 samples 11 were semantically related. In two samples which were not semantically related, ‘Sample 6’ had two revisions ‘305d4c1268b9, 3bf134f2c758’ related to ‘store mapping changes for settings’ but its third revision ‘9cdd0b2699b3’ was related to ‘store mapping for categories’. Additionally, their comments appeared to be semantically related but as per the code paths, it seemed that they were not actually related to each other. In the other ‘Sample 12’, the first revision ‘188f4243bald’ was related to the latest update of ‘fluent’ library and the second revision ‘8330d952235e’ was related to the latest update of ‘EntityFramework’ library and hence they were not semantically related.

In ‘SignalR’, out of 14 samples, 11 were semantically related. In three samples that were not semantically related, ‘Sample 8’ had two revisions related to self-host changes as per the comments but post analyzing the changed code a semantic relation could not be established in both the revision changes, hence we marked it as ‘MayBe’. In ‘Sample 3’, the revisions were done to remove unused code in multiple places but the author provided the same comments for both of them which resulted in high cosine similarity responsible for their grouping. In ‘Sample 14’, it had the fixes of the naming conventions identified by the static analysis tool named ‘FxCop’.

We have observed that our approach is able to semantically group change-sets which has the potential to enhance the identification of logical dependency with a margin of 20% error in detection. As per our sample analysis, we did not notice any instance of grouping by HashTag that is not semantically related, however the groups which were created

by semantic (cosine) similarity had few instances that were not related.

IV. RELATED WORK

Gall et al. are the first to introduce the concept of logical coupling [9] by analyzing the dependencies in 20 different product releases of a telecommunications switching system. D’Ambros et al. have made attempts to visualize logical coupling using an interactive visualization approach called Evolution Radar [7]. This approach was not composite and extensive enough for large software systems. Graves et al. [10] have shown that the future occurrence of faults can be easily predicted by the past revision histories of the software system. However, this reliability of this prediction can be questioned as it was never evaluated against a real-world evolving software product. Logical coupling has also been employed to predict changes in the software product [15] and was used to infer code decay [8]. There are many such use-cases which are introduced and practiced by software researchers. Mockus et al. [12] have found that the rate of widespread of a change over sub-systems and its related artifacts is a strong indicator for a presence of defects in a respective change. However, it requires a strong empirical validation on a real-world data set.

Ambros et al. [5] have reverse engineered a software system using an interactive visualization technique called the Evolution Radar, which can effectively break down the amount and complexity of the logical coupling information. Manishankar et al. [13] proposed new measurement for improving detection accuracy of evolutionary coupling by blending the concept location in a code-base to determine whether the changes to the co-changed entities are corresponding in nature and are thus related. Gustavo Ansaldi Oliva et al. [14] have proposed an approach to group timely-close and semantically-related change-sets containing the same author and commit message by using a sliding time window concept. In spite of such significant research being brought out by a variety of software practitioners, the semantic relationship between the artifacts was never made by linking the approach to detect logical coupling. The existing approaches are built under the assumptions that the developer will check-in all the related files in a single commit, whereas this is not a practical scenario. In contrast to existing state-of-art approaches, our approach is considerably different as it considers grouping of change-sets semantically in order to enhance the detection of logical coupling.

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a simple approach to group semantically related change-sets in atomic-commit featured Version Control Systems before performing logical coupling identification. We used Cosine Similarity and work item hash-tag match to group various change-sets. We were able to group almost 20% of the revisions. As part of our validation, we implemented the proposed algorithms in our approach to group various change-sets and presented preliminary evaluation results for the same. Our evaluation revealed promising results with a margin of error 20% in ‘Cosine Similarity’

grouping of change-sets. Whereas in ‘HashTag’ evaluation, we found that almost all the samples were semantically related. Intuitively, the results could easily be interpreted to conclude that ‘HashTag’ grouping provided a high degree of accuracy in grouping the change-sets, whereas ‘Cosine Similarity’ had few instances of false positives.

As part of our future work, we plan to detect hashtags by using pattern recognition techniques using a new algorithm that can reduce the possible HashTag patterns from the revision comments. We also plan to develop a technique to filter semantically unrelated files and exclude false positives. We observed that most of the false positive are related to ongoing design tasks “like refactoring, merge of branches”. Therefore we will plan to develop a filtering rule to exclude such change-sets as part of our future data sets. In regards to our present analysis, we have only targeted the main/trunk branch for detecting change-set groups. In the future, we would explore the possibilities to work with multiple branches. In regards to existing approaches, we ought to conduct an empirical study to understand the efficiency of our approach against the rest. We will be working further to enhance the effectiveness of our approach by evaluating it against large code-base.

REFERENCES

- [1] ngit. www.github.com/mono/ngit.
- [2] nopcommerce - asp.net open-source e-commerce shopping cart solution. www.nopcommerce.com.
- [3] Random sample calculator. www.raosoft.com/samplesize.html.
- [4] Signalr - incredibly simple real-time web for .net. www.signalr.net.
- [5] M. D. Ambros and M. Lanza. Reverse engineering with logical coupling. In *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on*, pages 189–198. IEEE, 2006.
- [6] H. S. Christopher Manning, Prabhakar Raghavan. *Introduction to Information Retrieval*. Camebridge University Press, 2008.
- [7] M. D'Ambros, M. Lanza, and M. Lungu. Visualizing co-change information with the evolution radar. 35(5):720–735, 2009.
- [8] S. Eick, T. Graves, A. Karr, A. Mockus, and P. Schuster. Visualizing software changes. 28(4):396–412, 2002.
- [9] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 190–198, 1998.
- [10] T. Graves, A. Karr, J. Marron, and H. Siy. Predicting fault incidence using software change history. 26(7):653–661, 2000.
- [11] A. Kumar. Modern information retrieval: A brief overview. *Bulliten of IEEE Computer Society Technical Committee on Data Engineering*, 4(24):35–43, November 2001.
- [12] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.
- [13] M. Mondal, C. K. Roy, K. Schneider, et al. Improving the detection accuracy of evolutionary coupling by measuring change correspondence. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 358–362. IEEE, 2014.
- [14] G. A. Oliva, F. Santana, M. Gerosa, and C. de Souza. Preprocessing change-sets to improve logical dependencies identification. In *Proceedings of the 6th International Workshop on Software Quality and Maintainability*, 2012.
- [15] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. 31(6):429–445, 2005.

A Robust Visual Tracker Based on DCF Algorithm

Menglei Jin, Weibin Liu*
Institute of Information Science
Beijing Jiaotong University
Beijing 100044, China
E-mail: wblu@bjtu.edu.cn

Weiwei Xing
School of Software Engineering
Beijing Jiaotong University
Beijing 100044, China

Abstract— Since Correlation Filter appeared in the field of video object tracking, it is great popular due to its excellent performance. The Correlation Filter based tracking algorithms are very competitive in terms of accuracy and speed as well as robustness. However, there are still some fields for improvement in the Correlation Filter based tracking algorithms. First, during the training of the classifier, the background information that can be utilized is very limited. Moreover, the introduction of the cosine window further reduces the background information. These reasons reduce the discriminating power of the classifier. This paper introduces more global background information on the basis of the DCF tracker to improve the discriminating ability of the classifier. Then, in some complex scenes, tracking loss is easy to occur. At this point, the tracker will be treated the background information as the object. To solve this problem, this paper proposes a novel re-detection component. Finally, the current Correlation Filter based tracking algorithms use the linear interpolation model update method, which cannot adapt to the object changes in time. This paper proposes an adaptive model update strategy to improve the robustness of the tracker.

Keywords—visual tracking; Correlation Filter; background modeling; re-detection and re-search; model update

I. INTRODUCTION

Video object tracking is a very important and popular study in the field of computer vision. It has applications in many areas, such as precision guidance in the military field, intelligent transportation and autonomous driving, and human-computer interaction. At present, more and more research results have been achieved in this field. Both tracking accuracy and speed have been greatly improved. However, it still faces many challenges, especially complex scenes such as lighting changes, motion blur, and occlusion [1]-[2].

We focus on single object tracking in this paper. At present, there are two types of methods in the field of object tracking. One is the generative approach, the other is the discriminative approach. It's not unfamiliar to generative models, such as classic mean-shift, particle filtering, kalman filtering and so on. Unlike the generative model, the discriminative model trains a classifier which can effectively distinguish between the object and the background. As far as the current research progress is concerned, the discriminative model is superior to the generation model.

The latest research progress of discriminative approach is Correlation Filter based algorithms. Correlation Filter is simply referred to as CF. Correlation is a concept in the field of signal processing. In real life, many problems can be solved with Correlation functions. It measures the degree of similarity of two signals at a time. Therefore, the more similar the two signals are, the larger the correlation value is. Been inspired by this idea, the MOSSE (Minimum Output Sum of Squared Error) tracker skillfully applies Correlation Filter to video object tracking [3]. Specifically, during the tracking process, we need to train a Correlation Filter template based on the object image block and desired output. When this template is applied to the candidate image blocks, the image block with the largest response value is regarded as the object. Why are Correlation Filter based algorithms capable of achieving so high speeds? This is mainly due to cyclic shift and Fourier transform, which can greatly reduce the computational complexity.

Nevertheless, there are still some problems to be solved in Correlation Filter based algorithms. (1) The problem of insufficient background information. It is well known that negative samples are very important in the training of classifier. However, in order to reduce the boundary effect caused by cyclic shift, the algorithm uses the operation of adding cosine window. This operation not only reduces the proportion of real samples, but also further reduces the background information. (2) The problem of tracking loss and model drift. In complex scenarios, tracking loss often occurs in the process of tracking. Traditional algorithms do not take measures to deal with this situation. (3) The problem of model updating cannot adapt to object changes. Classical algorithms often use fixed update rate to update the model linearly. This updating method has a great defect, it cannot reflect the change of the target in time. It can seriously affect tracking accuracy.

To solve these problems, this paper makes the following contributions. The proposed algorithm is based on DCF tracker. In order to improve the discriminant performance of the classifier, we combine the framework in CACF [4] with the DCF framework. Then, we add the operation of re-detection and re-search on the basis of the original algorithm flow. Here, re-detection depends on tracking confidence index. Finally, we introduce an adaptive model updating strategy, which can adapt to the appearance changes of the object in time. Experiments show that these measures are effective.

II. Related Work

Blome et al. first applied Correlation Filter to video object tracking. The MOSSE tracker uses an objective function with a minimum mean square error. Due to the full utilization of the

{Corresponding author: Weibin Liu, wblu@bjtu.edu.cn}
DOI reference number: 10.18293/SEKE2019-149

Fourier transform, the tracking speed of the algorithm is very fast. Its accuracy rate is 43.1% while maintaining 615fps. The disadvantage of the MOSSE tracker is that the feature representation is relatively simple, which greatly affects the tracking accuracy. The CSK (Circulant Structure with Kernels) tracker proposed by Henriques et al. solves the tracking problem from the perspective of machine learning [5]. It uses a cyclic shift sampling strategy to solve the problem of insufficient training samples that always exist in object tracking field. However, the shortcoming of the feature representation is still unresolved. Later, the KCF/DCF tracker was proposed [6]. It extends the feature representation of the image from the single channel gray features to multichannel HOG (Histogram of Oriented Gradients) feature. Compared with the CSK tracker, the tracking accuracy has increased from 54.4% to 73.2%. Among them, the difference between KCF (Kernelized Correlation Filter) and DCF (Discriminative Correlation Filter) is use of nuclear techniques. Similarly, the CN (Color Name) tracker extends the feature representation from gray features to color attributes [7]. The CN feature subdivides the RGB color into 11 common color in life [8]-[9]. To avoid high computational complexity, the algorithm uses the PCA method to process features. Later, Danelljan M et al. proposed the DSST (Discriminative Scale Space Tracker), which solves the multi-scale problem in object tracking [10]. During the tracking process, the algorithm designs two independent Correlation Filters to estimate the scale and the position of object. Another tracker that can solve multi-scale problem is SAMF (Scale Adaptive Filter with Feature Integration), which was proposed by Yang L. This algorithm combines the FHOG feature with the CN feature for the first time. It adopts a multi-scale detection algorithm in the field of object detection. Unlike the DSST tracker, the SAMF tracker [11] can simultaneously estimate the location and scale of the object.

At present, there are already some tracking algorithms that have done work on re-detection. The MOSSE tracker proposes a PSR indicator to measure tracking confidence. Then, LCT (Long-term Correlation Filter Tracker) introduces an additional Correlation Filter in the base of the DSST that can be used to detect object confidence [12]. If the tracking confidence factor does not meet the requirements, the algorithm initiates a research component. The algorithm uses the SVM classification algorithm to relocate. In terms of model updating, the LCMCF (Large margin tracking method with circulant feature maps) tracker proposes a robust model updating strategy [13]. The model is updated only when the tracking confidence coefficient reaches the historical mean. Otherwise, the model is not updated. In addition, the ECO tracker uses another more efficient way to perform model updates. The tracking model is updated every few frames [14].

III. OUR METHOD

In this section, we detail our proposed object tracking algorithm. Because the algorithm is based on the DCF tracker, we first introduced the basic tracker. Then, we introduced the core of this paper: global background modeling, re-detection component and adaptive model update strategy.

A. The Base Method: DCF tracker

The introduction of a large number of negative samples can improve the discriminative performance of the classifier during the tracking process. Many tracking algorithms determine the negative samples based on the distance of the object in the two frames. The overlap rate of these negative samples is actually very high. Therefore, the DCF tracker constructs training samples by cyclic shifting of the tracking object. As shown in Equation 1, the DCF tracker uses the regularized least squares classification algorithm to train the classifier.

$$\min_w \sum_i (f(x_i) - y_i)^2 + \lambda \|w\|^2 \quad (1)$$

Where f denotes a classification function, x_i and y_i represent training samples and expected outputs, respectively, W represents the weight of the classifier, and λ is a regularization parameter to prevent overfitting. Then we write it in the form of a matrix.

$$\min_w \sum_i (XW - y)^2 + \lambda \|w\|^2 \quad (2)$$

In formula 2, X and y represent the matrix form of x_i and y_i , respectively. To minimize the objective function, we take advantage of the nature of the cyclic matrix. The solution can be obtained by setting the derivative to 0.

$$\hat{w} = \frac{\hat{x}^* \odot \hat{y}}{\hat{x}^* \odot \hat{x} + \lambda} \quad (3)$$

Then, we introduce a kernel function to solve the problem of insufficient discriminant performance of the linear classifier. The kernel function maps the input data to a nonlinear feature space. Through the mapping, we can get the following expression:

$$w = \min_w \|\phi(X)w - y\|^2 + \lambda \|w\|^2 \quad (4)$$

Here, w satisfies $w = \sum_i \alpha_i \phi(x_i)$. Therefore, the above formula can be expressed as:

$$\alpha = \min_{\alpha} \|\phi(X)\phi(X)^T \alpha - y\|^2 + \lambda \|\alpha\|^2 \quad (5)$$

Through a series of derivations, the solution can be expressed as:

$$\alpha = F^{-1} \left(\frac{\hat{y}}{\hat{k}^{xx} + \lambda} \right) \quad (6)$$

Where k^{xx} is the first row of the kernel matrix k . The matrix k satisfies $k = \phi(X)\phi(X)^T$. In order to locate the tracking object in each subsequent frame, the DCF tracker also uses the cyclic shift to build the samples.

$$f(z_j) = w^T \phi(z_j) \quad (7)$$

z_j in Equation 7 represents a sample obtained by cyclically shifting the image block z , which satisfies $z_j = p^j z$. We can estimate the position of the object by calculating the detection values of all samples by the following formula.

$$\hat{f}(z) = (k^z)^T \alpha \quad (8)$$

Through the diagonalization operation, the above formula can be converted into:

$$\hat{f}(z) = \hat{k}^{xz} \odot \hat{\alpha} \quad (9)$$

Where k^{xz} is the nuclear correlation of x and z . Repeat this process and the tracking process is complete.

B. Global background modeling

In the tracking process of the KCF/DCF tracker, the search range is usually 1.5 times the size of the object. However, the operation of the cyclic shift is used in the construction of the sample, which brings the boundary effect. To alleviate the boundary effect, the DCF tracker adds a cosine window. However, the cosine window reduces not only the search range, but also the available background information available during the tracking process. In this paper, we combine the DCF framework with the framework in CACF to increase global context information.

During the training of the Correlation Filter, we sampled around the object. We train the ridge regression classifier using the object and the background image block. To get the closed solution, we regress the expected value of the global background image block to zeros. The objective function is:

$$\min_w \|A_0 w - y\|_2^2 + \lambda_1 \|w\|_2^2 + \lambda_2 \sum_{i=1}^k \|A_i w\|_2^2 \quad (10)$$

Where A_0 represents the cyclic matrix formed by the object image block, A_i represents the cyclic matrix constructed by the global background image block, and λ_1 and λ_2 are regularization parameters. In the objective function, the expected output value of the object image block is y . We recombine the object image block and the background image block to form a new matrix. This formula can be rewritten as:

$$f_p(w, B) = \|Bw - \bar{y}\|_2^2 + \lambda_1 \|w\|_2^2 \quad (11)$$

Here, B denotes a new matrix formed by combining the global background image block and the object, and \bar{y} denotes a label vector obtained by expanding the label y with 0. B and \bar{y} , satisfy,

$$B = \begin{bmatrix} A_0 \\ \sqrt{\lambda_2} A_1 \\ \vdots \\ \sqrt{\lambda_2} A_k \end{bmatrix} \quad \text{and} \quad \bar{y} = \begin{bmatrix} y \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (12)$$

Since $f_p(w, B)$ is a convex function, by deriving the objective function and setting the derivative to 0, a closed solution for w can be obtained:

$$w = (B^T B + \lambda_1 I)^{-1} B^T \bar{y} \quad (13)$$

Using the property of the cyclic matrix, the above formula can eventually be converted to:

$$w = [F \text{diag}(\hat{\alpha}_0^* \odot \hat{\alpha}_0 + \lambda_1 + \lambda_2 \sum_{i=1}^k \hat{\alpha}_i^* \odot \hat{\alpha}_i) F^H]^{-1} F \text{diag}(\hat{\alpha}_0^* \odot \hat{y}) \quad (14)$$

By performing a Fourier transform on w , we can get:

$$\hat{w} = \frac{\hat{\alpha}_0^* \odot \hat{y}}{\hat{\alpha}_0^* \odot \hat{\alpha}_0 + \lambda_1 + \lambda_2 \sum_{i=1}^k \hat{\alpha}_i^* \odot \hat{\alpha}_i} \quad (15)$$

The next step is object detection. The response map can be obtained by convoluting the learned filter with the image block z (search window) in the next frame. Much like the DCF tracker, the detection formula for solving the response map can be expressed as:

$$\hat{r}_d = \hat{z} \odot \hat{\alpha}_0^* \odot \hat{\alpha}_0 + \sqrt{\lambda_2} \sum_{i=1}^k \hat{z} \odot \hat{\alpha}_i^* \odot \hat{\alpha}_i \quad (16)$$

C. Re-detection and re-search components

In the traditional Correlation Filter tracking algorithms, the principle of tracking is to apply the Correlation Filter template to the detect image blocks, and regard the position of the peak in the response map as the position of the object. However, there is a drawback in such a detection method which the position with the largest response value does not always track the actual position of the object, especially in some complicated scenarios. Therefore, it is likely to bring about model pollution with using this method for position estimation directly, which affects the accuracy of the tracking algorithm. In this paper, we introduce a re-detection and re-search operation to improve this problem. We used the APCE indicator to determine the tracking performance of the current frame [13]. The APCE calculation method is as follows:

$$APCE = \frac{|\max(\phi(t)) - \min(\phi(t))|^2}{\text{mean}(\sum_{m,n} (g_{m,n} - \min(\phi(t)))^2)} \quad (17)$$

Where $\phi(t)$ represents the response map of the t -th frame. When the APCE indicator does not reach the historical average, the re-detection component is turned on. We re-search for the object at the position corresponding to the other peaks of the response map. Then, the response maps obtained by the correlation filter template at all candidate image blocks are respectively calculated. Finally, we determine the image block with the largest response value as the object.

D. Adaptive model update strategy

During the tracking process, the object often undergoes various changes. Therefore, the tracking algorithm needs to update the model in time to adapt to this change to improve the robustness of the tracking algorithm. This is very challenging. In the classic Correlation Filter based tracking algorithms, both KCF and DSST tracker use the following linear interpolation update method.

$$\hat{x}^n = (1 - \gamma) \hat{x}^{n-1} + \gamma \hat{x} \quad (18)$$

$$\hat{\alpha}^n = (1 - \gamma) \hat{\alpha}^{n-1} + \gamma \hat{\alpha} \quad (19)$$

Where n is the sequence number of the current frame, \hat{x} is the target representation model represented by the predicted position image block, and $\hat{\alpha}$ is the classifier parameter, γ represents the model update rate. Currently, the model update rate is determined by empirical values and is constant. The larger the value of γ , the faster the model update rate. Conversely, the smaller the value of γ , the slower the model update rate. When a large change occurs in the object, such as posture change and in-plane rotation, etc., γ should be selected as a larger value. When the tracking environment changes greatly and the tracking object does not change much, γ should be selected as a smaller value. However, when γ is too large, the update rate is too fast and it is easy to cause model drift. In summary, setting the model update rate to a fixed value is not appropriate because it does not accurately reflect changes in the object. Moreover, this update method easily leads to over-fitting of the model to several frames

of images. For example, the updated model is particularly sensitive to occlusion and deformation. In this paper, we propose an adaptive model update strategy. Here, we introduce a penalty coefficient ξ to control the model update rate. ξ is determined by the following formula.

$$\xi = \frac{\max(\phi(t))}{\max(\{\phi(1), \phi(2), \phi(3), \dots, \phi(t-1)\})} \quad (20)$$

IV. EXPERIMENT RESULTS

A. Dataset and evaluation index

In order to test the performance of the algorithm, this paper uses the OTB-2013 and OTB-2015 datasets, which are very popular in the field of video object tracking. Prior to the advent of OTB-2013, there was no recognized database here, so the tracking performance of the algorithm could not be accurately tested and compared. The significance of OTB-2013 is profound, which greatly promotes the development of tracking algorithms. OTB-2013 includes a total of 50 video sequences, while OTB-2015 extends it from 50 video sequences to 100 video sequences. The datasets contain 11 tracking difficulties: illumination changes, scale changes, occlusion, deformation, motion blur, fast motion, in-plane rotation, out-of-plane rotation, out-of-field, background clutter, low resolution. There are 26 gray sequences and 74 color sequences in 100 video sequences. The tracking targets that appear in OTB-2015 dataset include 36 entities, 26 faces/headers, and a total of 58897 frames. The length of the video sequence includes short-term and long-term, and the longest video sequence has more than 3,000 frames.

The OTB-2013 and OTB-2015 datasets provide two evaluation indicators: precision and success rate. Precision is an indicator used to measure the center pixel position deviation. As shown in Figure 1, the estimated tracking frame of the algorithm is S' , and its central pixel position is (x', y') . The actual tracking frame is S , and its center pixel position is (x, y) . Use the Euclidean distance when calculating the center pixel deviation. Typically, the center pixel deviation threshold is set to 20 pixels. The success rate is calculated as follows:

$$\phi = \frac{S \cap S'}{S \cup S'} \quad (21)$$

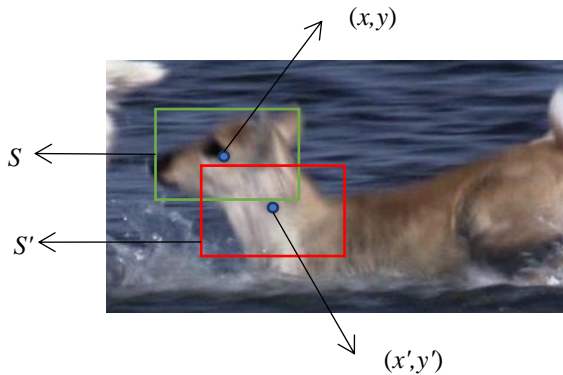


Figure 2. Ground truth and actual tracking results comparison chart

B. Parameter setting and algorithm flow chart

In practice, we implemented the proposed algorithm using the MATLAB language. In this paper, the kernel function type is set to a linear kernel, the image block padding is set to 2, λ_1 is set to $1e-4$, λ_2 is set to 26, and the linear interpolation factor for adaptation is set to 0.012. When calculating the HOG feature, the cellSize is set to 4 and the orientation is set to 9.

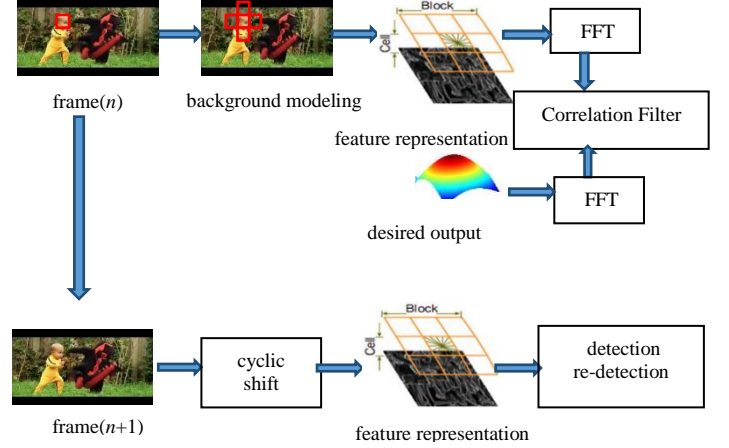
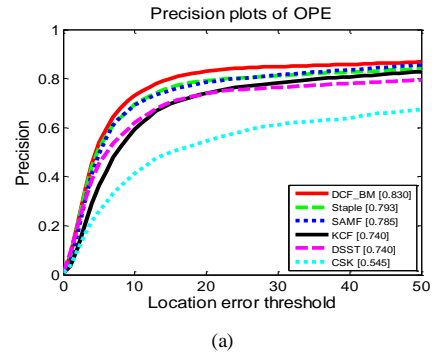


Figure 2. Algorithm flow chart

Figure 2 shows in detail the flow chart of the tracking algorithm proposed in this paper. Compared with the traditional Correlation Filter based algorithms, our proposed DCF_BM tracker has improved the training process of the Correlation Filter. On the negative samples of the training, not only the image blocks after the object cyclic shift are used, but also the image blocks around the object are also taken into consideration. In addition, an adaptive model update strategy is proposed in this paper to better adapt to object changes. Additionally, the re-detection and relocation components are also used in the DCF_BM tracker.

C. Performance testing and analysis

In this section, we compare the proposed algorithm with some of the Correlation Filter based trackers that have appeared in recent years. Because the KCF tracker has been significantly better than traditional trackers including Struck [15], TLD [16] and MEEM [17], we only compared algorithms include CSK, KCF, SAMF, DSST and Staple tracker [18].



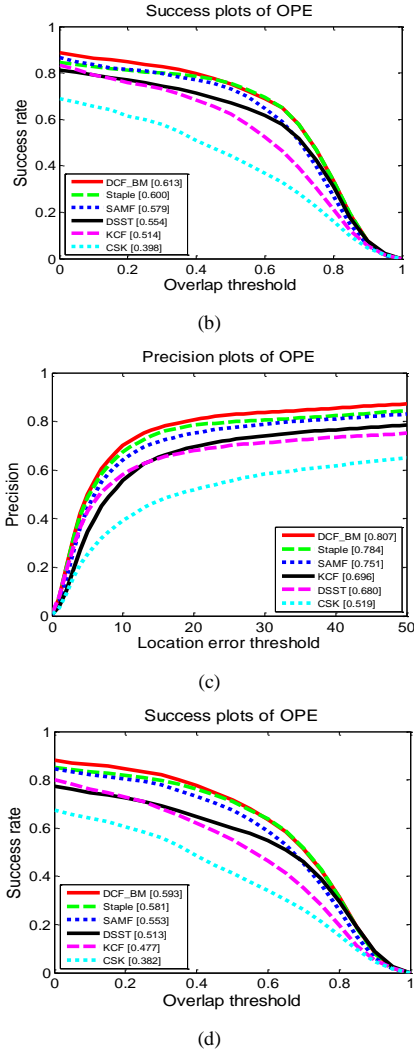


Figure 3. Experimental results on OTB-2013 and OTB-2015

The experimental results of these algorithms on the OTB-2013 dataset are shown in (a) and (b) of the Figure 3. It can be seen that the DCF_BM tracker proposed in this paper has obtained the optimal tracking result on the OTB_2013 dataset. The precision is 83.0% and the success rate is 61.3%. Compared to the baseline tracker, it achieves significant improvements in accuracy and success rate (both 9%). The experimental results on the OTB-2015 dataset are shown in Figures (c) and (d) of the Fig. 3. We can find that the DCF_BM tracker proposed in this chapter still achieves the best performance. It has an accuracy of 80.7% and a success rate of 59.3%. Compared with the classic KCF tracker, it has increased by 11.0% and 16.0% respectively in terms of accuracy and success rate. From these comparisons, one can conclude that the work done in this paper is effective and can improve tracking performance.

In addition, by comparing the experimental results, the Staple tracker ranks second in each result graph. This is mainly because the tracker combines HOG features and statistical color histograms. The response map is linearly weighted to obtain the final response map. The boundary effect is inherently a defect in the Correlation Filter based tracking framework, while the pixel-level color probability features are not affected by this factor. So, from this perspective, the DAT [19] in the Staple tracker

mitigates the boundary effect. However, the Staple tracker roughly uses the linear weighting method in the specific implementation. The Correlation Filter response occupies a weight of 0.7, which plays a major role, while the color probability response occupies a weight of 0.3, which plays a supporting role. However, this simple weighting method has an obvious disadvantage in that it cannot adaptively determine the weights of the correlation filter response and the color probability response.

In order to further verify the performance of the DCF_BM tracker proposed in this paper under different challenges, the attribute analysis on the OTB-2015 dataset was also carried out. The tracking performance of our proposed tracker and several other advanced trackers is shown in Tab. 1 and Tab. 2. It can be seen that the DCF_BM tracker proposed in this paper achieves the best performance in almost all attributes, in terms of precision or success rate, except LR (low resolution). In addition, the performance of the DCF_BM tracker is improved under various attributes compared to the standard KCF tracker, which is due to the use of background modeling.

Especially in occlusion (precision 73.8% vs. 65.1%) and fast motion (precision 74.7% vs. 62.1%) scenes, there are significant improvements in tracker's performance. It is mainly due to the application of re-detection and re-search components. When there is an abnormality in the response map, such as in fast motion and occlusion scenarios, we can try to expand the search area and re-detect the tracking object.

V. CONCLUSION

In this paper, a tracking framework based on DCF tracker is proposed. The introduction of global background information solves the problem that the background information in the traditional Correlation Filter framework is very limited, thereby improving the discriminating ability of the classifier. The DCF tracker adopts the model updating method of linear interpolation, which is difficult to adapt to the change of the object. The DCF_BM tracker proposed in this paper introduces a penalty coefficient that controls the update rate of the model, which significantly improves this shortcoming.

TABLE I. AVERAGE PRECISION SCORE ON OTB-2015 DATASET

attribute	DCF_BM	Staple	DSST	KCF	SAMF
IV(38)	0.817	0.791	0.721	0.719	0.715
OPR(63)	0.762	0.738	0.644	0.677	0.739
SV(64)	0.757	0.727	0.638	0.633	0.705
OCC(49)	0.738	0.726	0.597	0.630	0.726
DEF(44)	0.779	0.748	0.542	0.617	0.686
MB(29)	0.738	0.707	0.567	0.601	0.655
FM(39)	0.747	0.697	0.552	0.621	0.654
IPR(51)	0.799	0.770	0.691	0.701	0.721
OV(14)	0.708	0.661	0.481	0.501	0.628
BC(31)	0.785	0.766	0.704	0.713	0.689
LR(9)	0.638	0.631	0.567	0.560	0.685

TABLE II. AVERAGE SUCCESS SCORE ON OTB-2015 DATASET

attribute	DCF BM	Staple	DSST	KCF	SAMF
IV(38)	0.615	0.598	0.558	0.479	0.534
OPR(63)	0.549	0.534	0.470	0.453	0.536
SV(64)	0.537	0.525	0.468	0.394	0.495
OCC(49)	0.560	0.548	0.453	0.443	0.540
DEF(44)	0.566	0.554	0.420	0.436	0.509
MB(29)	0.567	0.546	0.469	0.459	0.525
FM(39)	0.571	0.537	0.447	0.459	0.507
IPR(51)	0.564	0.552	0.502	0.469	0.519
OV(14)	0.515	0.481	0.386	0.393	0.480
BC(31)	0.585	0.574	0.523	0.498	0.525
LR(9)	0.417	0.418	0.383	0.307	0.430

ACKNOWLEDGMENT

This research is partially supported by National Natural Science Foundation of China (No.61876018).

REFERENCES

- [1] Wu Y, Lim J, Yang M H. Object Tracking Benchmark[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2015, 37(9):1834-1848.
- [2] Wu Y, Lim J, Yang M H. Online Object Tracking: A Benchmark[C]// Computer Vision and Pattern Recognition. IEEE, 2013:2411-2418.
- [3] Bolme D S, Beveridge J R, Draper B A, et al. Visual object tracking using adaptive correlation filters[J]. 2010.
- [4] Mueller M, Smith N, Ghanem B, et al. Context-Aware Correlation Filter Tracking[C]// Computer Vision and Pattern Recognition. IEEE, 2017: 1387-1395.
- [5] João F. Henriques, Caseiro R, Martins P, et al. Exploiting the Circulant Structure of Tracking-by-Detection with Kernels[M]// Computer Vision - ECCV 2012. Springer Berlin Heidelberg, 2012.
- [6] Henriques J F, Caseiro R, Martins P, et al. High-Speed Tracking with Kernelized Correlation Filters[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015, 37(3):583-596.
- [7] Danelljan M, Khan F S, Felsberg M, et al. Adaptive Color Attributes for Real-Time Visual Tracking[C]// 2014 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2014.
- [8] Weijer J V D, Schmid C, Verbeek J. Learning Color Names from Real-World Images[C]// 2007 IEEE Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2007.
- [9] Van d W J, Schmid C, Verbeek J, et al. Learning Color Names for Real-World Applications[J]. IEEE Transactions on Image Processing, 2009, 18(7):1512-1523.
- [10] Danelljan M, Häger G, Khan F, et al. Accurate scale estimation for robust visual tracking[C]//British Machine Vision Conference, Nottingham, September 1-5, 2014. BMVA Press, 2014.
- [11] Li Y, Zhu J. A Scale Adaptive Kernel Correlation Filter Tracker with Feature Integration[J]. 2014.
- [12] Ma C, Yang X, Zhang C, et al. Long-term correlation tracking[C]// Computer Vision and Pattern Recognition., 2015: 5388-5396.
- [13] Wang M, Liu Y, Huang Z. Large Margin Object Tracking with Circulant Feature Maps[J]. 2017:4800-4808.
- [14] Danelljan M, Bhat G, Khan F S, et al. ECO: Efficient Convolution Operators for Tracking[J]. 2016.
- [15] Hare S, Saffari A, Torr P H S. Struck: Structured output tracking with kernels[J]. 2011.
- [16] Kalal Z, Mikolajczyk K, Matas J. Tracking-Learning-Detection.[J]. IEEE Trans Pattern Anal Mach Intell, 2012, 34(7):1409-1422.
- [17] Zhang J, Ma S, Sclaroff S. MEEM: Robust Tracking via Multiple Experts Using Entropy Minimization[M]// Computer Vision - ECCV 2014. Springer International Publishing, 2014:188-203.
- [18] Bertinetto L, Valmadre J, Golodetz S, et al. Staple: Complementary learners for real-time tracking[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016: 1401-1409.
- [19] Possegger H, Mauthner T, Bischof H. In defense of color-based model-free tracking[C]// Computer Vision and Pattern Recognition. IEEE Computer Society, 2015.

A Novel Algorithm for Exemplar-based Image Inpainting

Yaru Cheng, Weibin Liu *

Institute of Information Science
Beijing Jiaotong University
Beijing 100044, China

Corresponding author: Weibin Liu, wblu@bjtu.edu.cn

Weiwei Xing

School of Software Engineering
Beijing Jiaotong University
Beijing 100044, China

Abstract—In traditional exemplar-based image inpainting algorithm, the confidence value will rapidly decrease to zero as the inpainting process progresses. As a consequence, it will lead to unreliable result of the priority calculation and wrong direction of the process. In addition, traditional methods usually use the sum of squared differences (SSD) criterion to search the optimal matching block. Since the matching criterion is single and the precision is limited, the process is easy to produce mismatch. In order to solve the above problems, an improved algorithm has been proposed in this paper. First, we proposed a new confidence update algorithm through replacing the previous linear function form by using a logarithmic function form, which can suppress the phenomenon that the confidence attenuation is too fast and improve the accuracy of guiding and inpainting direction. Then, we combine the physical distance between blocks and traditional SSD matching criterion to improve matching accuracy. The experimental results show that the algorithm overcomes the shortcomings of the traditional algorithm and provides higher quality image restoration effects and better visual effects.

Keywords—Exemplar; image inpainting; matching accuracy; Confidence term update

I. INTRODUCTION

Digital image inpainting technology has broad application prospects and is a prerequisite for image compression, image enhancement, image recognition and other technologies [1]. First, it need to research and solve how to better detect the damaged part of the image. Then, image inpainting algorithm can automatically inpaint the image according to the known information around the damaged area. Therefore, the research and exploration of image inpainting technology is a subject including both theoretical and practical value [2]. In the time, the requirements for images and video quality around us become higher and higher [3]. Therefore, it is particularly important to research on digital image inpainting.

Recently, many researchers have proposed effective image inpainting models. According to the inpainting theories, they can be divided into two categories: geometry based approaches and exemplar-based approaches [5]. Image inpainting techniques based on non-texture structures are often used to inpaint images with small damaged areas, such as scratch repair and text contamination repair [6]. In 2000, Bertalmio et al proposed BSCB model based on partial differential equation image inpainting [7]. The main idea of this model is to extending the isophote line of the damaged region's boundary and propagating

it into the damaged area. Although the BSCB model has a good inpainting result, it is based on the diffusion and transmission of the inpainting process. Objectively, it is very difficult to carry out mathematical analysis. Chan and Shen proposed the total variational model (TV model) [8] and the curvature-driven diffusion model (CDD model) [9] based on the BSCS model. Then, Telea proposed a fast matching algorithm (FMM) [10].

In 2004, Criminisi proposed the exemplar-based inpainting algorithm, which is on patch level [11]-[12]. Then, many exemplar-based image inpainting methods have emerged [13] and they have been widely used. In [14], Barnes proposed a new image editing algorithm PatchMatch to find approximate nearest-neighbor patches from image patches. This interactive technique has been used by Adobe photoshop because of the flexibility, convenience and simplicity. In [15], Liu used multiscale graph cuts in the process. In [16], Komodakis used MRF-based global optimization method to solve the inpainting problem, which is known as priority belief propagation (p-BP). There are some other researchers do some work based on priority belief propagation [17]-[18]. Then, Ruzic introduced context-aware patch-based image inpainting, where candidate patches are searched over the entire source region based on contextual similarity [19]. The author found a new priority calculation in [20] and make up for the shortcomings in Criminisi method. In [21], they used Pixel Inhomogeneity Factor (PIF) to replace the data term. In [22], Huang use the planar structure guidance to solve the inpainting problems. In [23], Deng separated the confidence term and data term, then he proposed an automatic algorithm to estimate steps of the new priority definition.

This paper proposed a novel algorithm for exemplar-based inpainting method. The main improvement is to change the matching criteria and confidence update function. First, we propose an improved confidence update function to maximize the accuracy of the guided inpainting direction. Then, we combine the physical distance between blocks and traditional SSD matching criterion to search the best matching patch. Novel method overcomes the drawback of Criminisi algorithm in visual inconsistency. We compare the method with two methods in some structure and texture pictures.

II. RELATED WORK

In this section, we mainly introduce the exemplar based inpainting method [12]. First, the image is divided into two parts: the target region is defined as the area which is to be inpainted,

{Corresponding author: Weibin Liu, wblu@bjtu.edu.cn}

DOI reference number: 10.18293/SEKE2019-152

using Ω to represent. The left area is called source region Φ . Additionally, $\delta\Omega$ represents the boundary of the target region. The inpainting task is to fill in the target region Ω using the image information from the source region Φ . The inpainting process is shown in Fig.1.

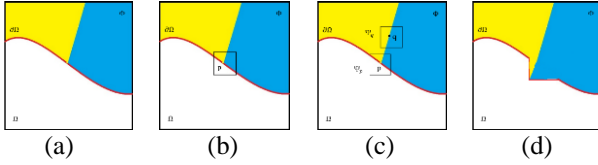


Fig.1 Inpainting process by Criminisi Method [12]

A. Deciding the filling order

This step is mainly aimed to determine the filling order of the patch in the target region. For each pixel p along the target region's boundary, we set a square patch φ_p which is centered on the pixel. In our paper, we set the patch size with 9×9 pixels throughout the process. After computing the priorities of all pixels along the boundary, we choose the pixel p with the highest priority as the center pixel, then the patch φ_p will be the target patch to be inpainted.

In Criminisi method [12], the filling priority is defined as follows:

$$P_{(p)} = C_{(p)} D_{(p)} \quad (1)$$

Where $C_{(p)}$ represents the confidence term and $D_{(p)}$ represents the data term and they are calculated by the following equations:

$$C_{(p)} = \frac{\sum_{q \in \varphi_p \cap \bar{\Omega}} C_{(q)}}{|\varphi_p|}, \quad 0 \leq C_{(p)} \leq 1 \quad (2)$$

$$D_{(p)} = \frac{|\nabla I_p^\perp \cdot n_p|}{\alpha}, \quad 0 \leq D_{(p)} \leq 1 \quad (3)$$

In equation (2), $\bar{\Omega}$ denotes the complementary set of target region Ω , $|\varphi_p|$ represents the area of the patch φ_p . During initialization, the function $C_{(p)}$ is set to

$$C_{(p)} = 0, \quad \forall p \in \Omega \quad (4)$$

$$C_{(p)} = 1, \quad \forall p \in \bar{\Omega} \quad (5)$$

For each pixel p on the boundary of target region, the confidence term $C_{(p)}$ equals to the ratio between the sum of pixels confidence in $\varphi_p \cap \bar{\Omega}$ and the total number of nonzero elements in φ_p . The higher of $C_{(p)}$, the more reliable information surrounding p . The intention is that we can inpaint the patch with more reliable information firstly.

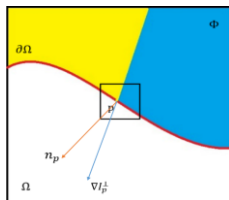


Fig. 2 The iron of n_p and ∇I_p^\perp

Similarly, n_p is the unit vector orthogonal to the front $\delta\Omega$ in the pixel p and ∇I_p^\perp is the isophote vector (where \perp denotes the orthogonal operator). $D_{(p)}$ represents the strength of isophotes hitting the front $\delta\Omega$ at each iteration. Wherever, α is the

normalization factor. Data term guarantees that the picture is inpainted following by the isophotes, which will make sure the linear structure is first filling. Fig. 2 show the details of $D_{(p)}$.

B. Select the best matching patch

This step mainly does searching work whose aim is to find the best matching patch $\varphi_{\hat{q}}$ in the source region for the target patch φ_p whose priority is the highest. Equation (6) is the measurement of the similarity between each patch φ_q in the source region and the target patch φ_p :

$$\varphi_{\hat{q}} = \arg \min_{\varphi_q \in \Phi} d(\varphi_{\hat{p}}, \varphi_q) \quad (6)$$

$$d_{SSD}(\varphi_p, \varphi_q) = \sum \left[\begin{aligned} & (R_{\varphi_p} - R_{\varphi_q})^2 + \\ & (G_{\varphi_p} - G_{\varphi_q})^2 + (B_{\varphi_p} - B_{\varphi_q})^2 \end{aligned} \right] \quad (7)$$

The distance $d(\varphi_{\hat{p}}, \varphi_q)$ is defined as the sum of squared differences (SSD) of the known pixels in the two patches. The details is shown in equation (7) where R , G and B represents the value of intensity of each color channel.

C. Inpaint the unknown area and update the confidence term

In this step, the algorithm fills the unknown area by putting the best matching patch $\varphi_{\hat{q}}$ to the target patch φ_p in the corresponding region. After that, we need to renew the boundary of the target region $\delta\Omega$ and update the confidence term using the following equation:

$$C(q) = C(p), \quad \forall q \in \varphi_p \cap \Omega \quad (8)$$

Then, just continue to run the above steps until the unknown region is fully inpainted.

III. PROPOSED METHOD

In this section, the new update function for confident term and a new searching method are proposed.

A. Confidence term updating

Updating confidence term is an important step in Criminisi algorithm. Once the patch φ_p is filled, the unknown pixel becomes a known point which will result the confidence value changing. Criminisi algorithm simply uses the best matching patch to inpainting φ_p , and replace confidence term of the center pixel p with the corresponding pixel's confidence. It is equivalent to using the function $f(x) = x, x \in [0, 1]$ to update the confidence value. Along with the iterative process progresses, the confidence value will rapidly decrease and tend to zero, which may result in an incorrect inpainting direction.

In order to suppress the decay of confidence term in traditional methods, this paper proposes a new confidence update function. We suppose the function $h(x)$ as the following:

$$h(x) = \log_2(x + 1) \quad (9)$$

Using the improved confidence function, the update equation used in the confidence update phase will be the following:

$$C(q) = h(C(p)) = \log_2(C(p) + 1) \quad (10)$$

Where $\forall q \in \varphi_p \cap \Omega$.

B. The new matching method

The Criminisi algorithm uses SSD criteria in the matching processing to determine the similarity between sample blocks. Just calculate the sum of the squares difference of all known pixels in the block, and then select a target with the smallest error from the current repaired sample block as the best match in the source region. The matching formula is

$$\varphi_{\hat{q}} = \arg \min_{\varphi_{q \cap \Phi}} d(\varphi_{\hat{p}}, \varphi_q) \quad (11)$$

Since the matching principle is too singular, and only the color feature is considered. When adopt the global search method, multiple optimal sample blocks may appear at a distance from the area to be repaired. Then the algorithm randomly selects one as the matching patch. Because the algorithm is a serial repair, the error copy of a patch will affect the subsequent match, which is easy to cause error accumulation and the quality of repair is degraded.

In an image, it can be divided into several regions according to different structural features. For a general candidate to be matched, the matching block should appear in the same or similar region as its own structure. And the probability that the matching block appears in other regions is small. Sometimes even if such a situation occurs, it may be a mismatch. This is because the information of the adjacent space is highly correlated, and the pixels that are far apart are less correlated, so the probability that the matching patch appears in the neighborhood is the largest. In order to minimize the probability of mismatching, we proposes an improved patch matching principle that incorporates the distance between the candidate patch center point and the target patch center point into the original similarity measure function.

When searching for matching patches, we still choose global search. Because global search can overcome the shortcomings of local search's greedy character. So it can select the better matching patch to the greatest extent, and the target block inpaint can achieve global optimization. In the global search, we combine the SSD and physical distance for the calculation. The new matching formula is as follows:

$$d_{new}(\varphi_p, \varphi_q) = d_{SSD}(\varphi_p, \varphi_q) * e^{dis(p,q)} \quad (12)$$

In our new calculation method, we combine SSD and physical distance to find the optimal matching block. In order to avoid that the physical distance is zero, we use the exponential form of the physical distance to calculate. The experimental results show that the new matching criterion significantly reduces the false matching rate, and the repair results are more in line with the visual connectivity of the human eye.

IV. EXPERIMENT AND ANALYSIS

In the section, we test the proposed method on a few images with various background. Compared with the classical and state-of-art methods mentioned above, there are some improvements that can be seen from our results. Furth ermore, we set the same size for the square patch φ_p in the experiments for Criminisi's method, Deng's method and the proposed method for the fairness.

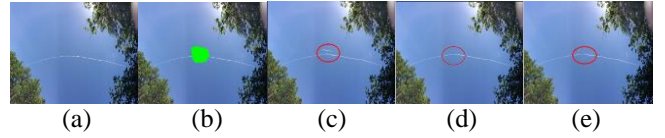


Fig. 3 Sky with a curve (a) original image (b) region need to inpainting (c)result made by[12] (d)result made by [23] (e) result made by proposed method

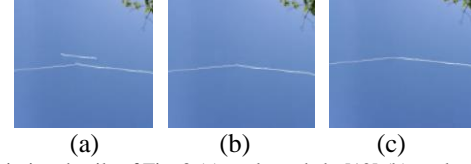


Fig. 4 Inpainting details of Fig. 3 (a)result made by[12] (b)result made by [23] (c)result made by proposed method

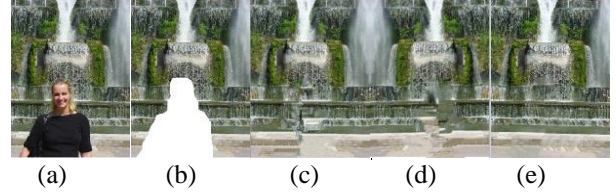


Fig. 5 Remove the girl(a) original image (b) region need to inpainting (c)result made by[12] (d)result made by [23] (e) result made by proposed

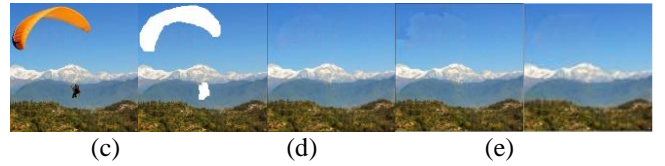


Fig. 6 Remove the Paragliding (a) original image (b) region need to inpainting (c)result made by[12] (d)result made by [23] (e) result made by proposed method

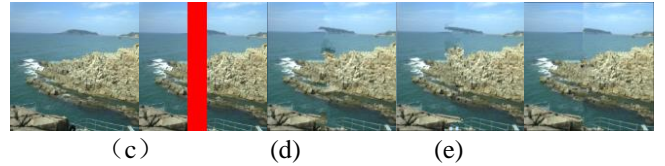


Fig. 7 Inpaint the red area in the sea(a) original image (b) region need to inpainting (c)result made by[12] (d)result made by [23] (e) result made by proposed method

Fig. 3 is the repair of a sky image. Obviously, a short line is emerged on the upper side of the curve in the result of Criminisi's method. Both Deng's algorithm and our algorithm avoid such mis-inpainting, and the results are more in line with real-life performance and human vision. The details of the inpainting results are shown in Fig. 4, which is more vivid to observation. In detail of the results, our method is much smoother than Deng's in terms of visual effects.

Fig. 5 is the image used in the Criminisi's paper. As can be seen from the results, the algorithm proposed by us has been greatly improved. Both competitive in terms of structure and texture. At the same time, we can see the effectiveness of the new algorithm in the subsequent of PSNR value. Fig. 6 is a target object removal for an image. The results are not clearly distinguishable. For this, we can use PSNR to judge. In the PSNR comparison in Table 1, we can observe that their values have not much difference, but there are still some. It can be seen that our algorithm is relatively better.

Fig. 7 is a representation of a seaside picture of a selected area. As can be seen from the results, the method we proposed on the repair of the coast is the best, and the results of the other two methods have spit out. Our results look smoother and more in line with real-life coastal conditions. Our method did not show particularly good results in the inpaint of the reef in the middle of the sea. In addition, in the three inpaint results, we are very clear to see the occurrence of fault phenomenon. This shows that we still have a lot of research space to solve the problem.

PSNR (peak signal-to-noise ratio) is an objective standard widely used for evaluating the quality of image processing. In order to evaluate the effect of the inpainted image, the PSNR value is usually used to measure the inpainting performance of the algorithm. The larger the PSNR value is, the smaller the degree of image distortion. Thus, the better the algorithm inpainting performance. PSNR is calculated as:

$$\text{PSNR} = 10 \log \left\{ \frac{255^2}{\frac{1}{M \times N} \sum_{i=1}^M \sum_{j=1}^N [f(i,j) - \hat{f}(i,j)]^2} \right\} \quad (12)$$

This paper also uses the PSNR to evaluate the effect of inpainting performance. Some PSNR values are shown in Table 1. It can be seen from Table 1 that the image inpainting methods proposed in this paper can obtain better results for images of different damaged regions. Since the structure and texture of the area to be repaired in Fig. 7 are relatively simple, the PSNR values of their results are similar regardless of the method. For images with more complex textures, as shown in Fig 6 and 8, the difference in PSNR values will be larger. Criminisi algorithm has difficulty in determining the confidence in the priority calculation and it is difficult to search for the best matching block, so that the PSNR value of the algorithm result is the lowest. Although Deng's algorithm improves the priority calculation, there is still problems in selection of the best matching block, which makes the PSNR value of the algorithm's result middle. Our method not only improves the confidence update, the selection of the best matching patch is strengthened. So, the PSNR value of our method gets the highest.

TABLE I.	Comparison of PSNR Value for different methods		
	Criminisi	Deng	ours
Sky	44.6362	46.9447	47.6480
Girl	13.8362	13.9845	14.0606
Paragliding	17.1514	17.1829	17.1924
Sea	25.7422	24.2435	26.9701

V. CONCLUSION

This paper proposes a new confidence update function and the new matching method. The new confidence update function can effectively reduce the phenomenon that the confidence attenuation is too fast, and improve the accuracy of the image guidance repair direction. At the same time, combining the physical distance between blocks and traditional SSD matching criterion as the new matching method effectively improves the matching accuracy. The proposed method obtains competitive results when handle pictures with strong structure or texture compared with other state-of-the-art inpainting methods. Nevertheless, our method performed not so well when the picture including both strong structure and texture.

ACKNOWLEDGMENT

This research is partially supported by National Natural Science Foundation of China (No.61876018).

REFERENCES

- [1] P. Buysse, M. Daisy, D. Tschumperle, and O. Lezoray. Exemplar-based inpainting: Technical review and new heuristics for better geometric reconstructions. *IEEE Trans. Image Processing (TIP)*, 24(6), 2015
- [2] C. Guillemot and O. Le Meur, "Image inpainting: Overview and recent advances," *IEEE Signal Process. Mag.*, pp. 127–144, Jan. 2014
- [3] Chan T.F, Shen J. Mathematical models for local nontexture inpainting. *SIAM Journal of Applied Mathematics*. 2002; 62:1019–1043.
- [4] Freeman W.T, Jones T.R, Pasztor E.C. Exemplar-based super-resolution. *IEEE Computer Graphics and Applications*. 2002; 22:56–65.
- [5] Liang L, Liu C, Xu Y, Guo B, Shum H.Y. Real-time texture synthesis using patch-based sampling. *ACM Trans on Graphics*. 2001; 20:127–150.
- [6] A. Efros and T. Leung, "Texture synthesis by non-parametric sampling," in *Proc. Int. Conf. Computer Vision (ICCV)*, Sept. 1999, pp. 1033–1038.
- [7] Bertalmio M, Sapiro G, Caselles V, Ballester C. Image inpainting. *Proc ACM SIGGRAPH Computer Graphics (SIGGRAPH)*. 2000; p. 417–424.
- [8] Chan T, Shen J (2001) Local inpainting models and tv inpainting. *SIAM J Appl Math* 62(3):1019–1043.
- [9] Chan T, Shen J (2001) Non-texture inpainting by curvature-driven diffusions. *J Vis Commun Image Represent* 4(12):436–449.
- [10] Telea A (2004) An image in-painting technique based on the fast marching method. *J Graph Tools* 9(1):23–34
- [11] Criminisi A, Perez P, Toyama K. Object removal by exemplar-based image inpainting. *Proc IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2003; p. 721–728.
- [12] Criminisi A, Perez P, Toyama K. Region filling and object removal by exemplar-based image inpainting. *IEEE Trans Image Processing*. 2004; 13:1200–1212
- [13] Qian Fan, Lifeng Zhang. A novel patch matching algorithm for exemplar-based image inpainting. *Multimed Tools Appl*(2017)
- [14] Xu Z, Sun J. Image inpainting by patch propagation using patch sparsity. *IEEE Trans Image Processing*. 2010; 19:1153–1165
- [15] Liu and V. Caselles, "Exemplar-based image inpainting using multiscale graph cuts," *IEEE Trans. Image Process.*, vol. 22, no. 5, pp. 1699–1711.
- [16] N. Komodakis and G. Tziritas, "Image completion using efficient belief propagation via priority scheduling and dynamic pruning," *IEEE Trans. Image Process.*, vol. 16, no. 11, pp. 2649–2661, Nov. 2007.
- [17] J. Mukherjee, and S. K. D. Mandal, "Image inpainting through metric labeling via guided patch mixing," *IEEE Trans. Image Process.*, vol. 25, no. 11, pp. 5212–5226, Nov. 2016.
- [18] Barnes C, Shechtman E, Finkelstein A, Goldman D.B. PatchMatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (TOG)*. 2009; 28
- [19] T. Ruzic and A. Pizurica, "Context-aware patch-based image inpainting using Markov random field modeling," *IEEE Trans. Image Process.*, vol. 24, no. 1, pp. 444–456, Jan. 2015.
- [20] W.H. Cheng, et al., Robust algorithm for exemplar-based image inpainting, *The International Conference on Computer Graphics, Imaging and Vision (CGIV 2005)*, 2005, pp. , 64–69
- [21] Q Fan, H Liu, Z Fu, X Li, Exemplar-Based Image Inpainting Based on Pixel Inhomogeneity Factor; *Proceedings of APSIPA Annual Summit and Conference 2017*, 12 - 15 December 2017, Malaysia
- [22] J. Huang, S. Kang, N. Ahuja, J. Kopf, Image completion using planar structure guidance, *ACM Trans. Graphics* 33 (4) (2014) 129. github.com/jbhuang0604/StructCompletion
- [23] Deng L-J, Huang T-Z, Zhao X-L (2015), "Exemplar-Based Image Inpainting Using a Modified Priority Definition." *PLoS ONE* 10(10): e014119

Trajectory Similarity Computation based on Interpolation and Integration

Zengwei Zheng

School of computing and Computational Science
Zhejiang University City College
Hangzhou, China
zhengzw@zucc.edu.cn

Wenwang Chen

School of Computer Science and Technology
Zhejiang University
Hangzhou, China
wwchen@zju.edu.cn

Yuanyi Chen

School of computing and Computational Science
Zhejiang University City College
Hangzhou, China
chenyuanyi@zucc.edu.cn

Dan Chen

School of computing and Computational Science
Zhejiang University City College
Hangzhou, China
chend@zucc.edu.cn

Abstract—Trajectory similarity computation is one of the most fundamental functionality, which is applied in many fields, such as trip trajectory mining to find the most popular routes and similar ones, and identify the routes of animal migration and even the stock trends. There are two different kinds of search thoughts, the advanced deep learning method and traditional points matching methods. However, the exiting methods are not totally perfect to solve the trajectory similarity computation problem. The deep- learning method has original problem that it needs a large size of dataset resulting in the requirement of the training time much more than we expected. While the traditional points matching method often suffer from noise and non-uniform sampling rates, because points matching often treats it as two different sequences when the unequal points turn up. In other word, it is often sensitive to the noise which lowers the correct rate of the similarity computation. Based of the statement upon, we propose a new method — applying the interpolation and deformed integration to similarity computation. Experiments shows that our method is robust to the noise and non-uniform sampling rate.

Keywords—*interpolation; deformed integration; robust; similarity computing; self-similarity;*

I. INTRODUCTION

With the development of the GPS-enabled devices, trajectory data is being collected over time. In other word, the way we get trajectory data is more and more accessible. Generally, a trajectory is often represented by a sequence of discrete points or locations. And we can map it to a spatial domain, e.g., 2D Euclidean space, which leads to much many of the trajectory similarity computation methods. Computing similarity between two trajectories is fundamental functionality for many applications, such as the migration patterns of animals, identifying hot routes in cities to avoid traffic jam, recommending popular trip routes for travelers and trajectory clustering. As a classic computation problem, a large amount of

traditional methods have been proposed, such as longest common sub-sequence (LCSS) [2], edit distance on real sequences (EDR) [4], and dynamic time warping (DTW) [1]. Besides, recently, as the deep learning develops, a method related--t2vec [6], inspired by word2vec, had been proposed.

In most situation, the traditional methods are trying to make points pair match to minimize their distance. But such operation would suffer from some problems, such as the non-uniform sampling rates and noise. For example, due to the devices' inherent limitation, it may fail to grasp the exact location information. What's more, when bypassing the rough area, it is hard for the devices to locate the accurate position. They all cause the generation of the noise. And the devices may lower the frequency of recording out of energy consideration, which leads to the non-uniform and low sampling rates. To cope with the situation, deep-learning-based method had been proposed and achieved remarkable improvement. However, deep learning itself has inherent defect. It needs a large size of dataset. And the training time and the running time is often required much more than our expectation. To deal with the problems of non-uniform sampling rates, noise and low efficiency, we propose a new method — applying interpolation and deformed integration to trajectory similarity computation. Other than the points matching method, we choose the curve fitting to further research the similarity of different trajectories. The experiment shows us that our method is full of vitality, and it makes noteworthy improvement in both accuracy and efficiency. Overall, the paper makes following contributions:

1) We proposed a trajectory similarity computation model based on the curve fitting. And our method is robust to the non-uniform sampling rates and noise, and computes the similarity in linear time (we will discuss it in the later part).

2) We conduct extensive experiments to verify our method outperforming the traditional points matching method in both accuracy and efficiency.

3) We also compare our method with the traditional ones under different proportions of test set and label set.

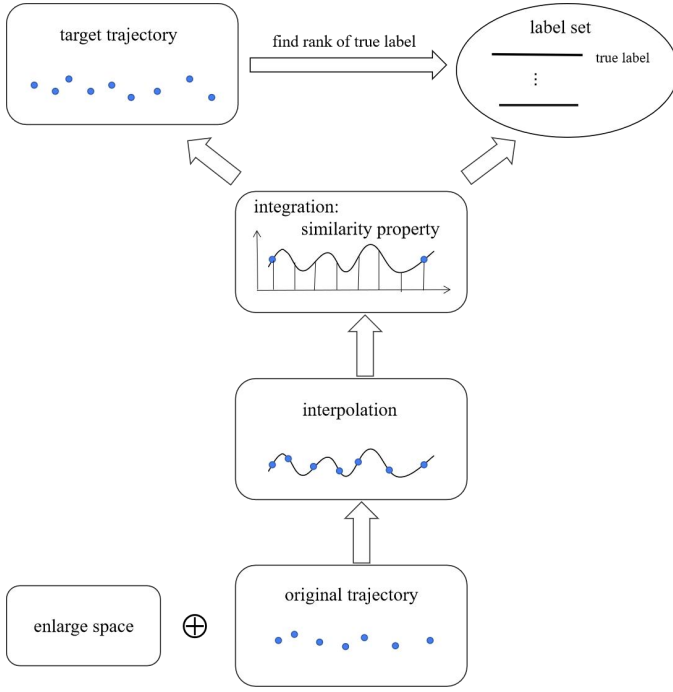


Fig. 1: Framework of proposed method

II. RELATED WORK

We briefly review the related work on the trajectory similarity computation. Computing the similarity, or say, the distance between two different trajectories is fundamental functionality in many computing tasks. And it is obvious that the accuracy and the efficiency is what we spot on. DTW [1] was first attempt at tackling the local time shift issue for computing trajectory similarity. T2vec [6] exploits deep representation learning to learn a vector inputted to the encoder-decoder framework to get a trained deep learning model. It highly increased the accuracy of the trajectory similarity computation, but being weak in the efficiency, which is left to be solved. In fact, not only t2vec method, almost every deep learning methods such as vRNN[8] share the same inherent defect. They need a large amount of data to train a model, which inevitably results in the long running time. And in some time, the loss of efficiency is unaffordable. APM[7] is another deep learning method. Despite the training time, it is worth mentioning that it solves the issue by learning transition patterns of anchor points from historical trajectories. EDR [4] was proposed to further deal with the problems of the spatial semantics in trajectories. Z. Chen et al. [15] proposed a kind of improved Frechet Distance to deal with the problem of the spatial sequences similarity computation to identify the most popular routes. To improve the robust to the variation in the sampling rates, EDwP[10] was proposed. What's more, EDwP uses the linear interpolation to compute the least loss of the insertion operation, which makes this method identical and inspires our computing measure. And ERP[3] and the model-driven approach MA[18] were proposed to better capture the spatial features in trajectories. They present different models

to speed the proceeding of capturing compared to other methods separately. Wang et al. [13] researched the effect of these similarity computation methods according to their robustness to noise and varying sampling rates.

Moreover, most of the aforementioned existing traditional measures for trajectory similarity computation has time complexity $O(n^2)$, while our method has the linear time complexity $O(n+v)$, where v represents the average length of trajectories, to measure the similarity, which reflects on the experiment -- our method is much more faster than others.

III. PROPOSED METHOD

In this part, we discuss the fundamentals of the proposed method. And then, we will prepare the preliminaries for the experiments. The method's framework is seen in Fig. 1.

A. Interpolation

Interpolation is a kind of curve fitting method to find a polynomial curve to best fit the discrete points. The definition of the interpolation: in an interval $[a, b]$, given n discrete points (x_k, y_k) , $k = 1, 2, \dots, n$, for every x , we need to find the corresponding y . $f(x)$ is defined in $[a, b]$, and

$$f(x_k) = y_k \quad k = 1, 2, \dots, n \quad (1)$$

And now, what we need to do is to find $g(x)$, for the n discrete points,

$$g(x_k) = f(x_k) \quad k = 1, 2, \dots, n$$

We call $g(x)$ an interpolation of $f(x)$ in $[a, b]$. In most cases, polynomial curve is applied when the fitting operation happens. Because for the given discrete points, the polynomial curve is much more convenient to conduct. And the sine-like polynomial curve is more stable because it usually approaches the reality than other fitting methods.

There are some reasons we adopt the interpolation instead of other fitting methods. According to the definition, we can get that the interpolation would absorb all the given discrete points without abandoning any of them. In our experiments, the most of trajectories in the dataset are collections of only dozens of latitude/longitude coordinates. For a complete route, each of them is needed. What's more, one of later operations is to distort some of the points of a trajectory (we will talk about it in detail lately) to simulate the noise in reality. That means it may destroy the simulated environment, or make the excessive operation to desert some points for the fitting.

So in this paper, we adopt *Lagrange Interpolation*, i.e., a sort of polynomial interpolation. In mathematics, Lagrange interpolation finds a polynomial that happens to take the observed values at each observed point. That means, it can provide a polynomial function that happens to cross all the given points in a two-dimensional plane.

Theorem *Polynomials that satisfy the interpolation condition (1) and whose degree does not exceed n exist and unique.*

The theorem can be proved in the establishment procedure of the Lagrange polynomial. Let set D_n be the collection of angular coordinates of the points (x_k, y_k) ,

$k=1,2,\dots,n$, $D_n=\{1,2,\dots,n\}$, there are n polynomials $p_j(x)$, $j \in D_n$, $\forall k \in D_n$,

$$p_k(x) = \prod_{i \in B_k} \frac{x - x_i}{x_k - x_i}$$

Where $B_k = \{i \mid i \neq k, i \in D_n\}$. And it is obvious the degree of $p_k(x)$ is $n-1$. Finally, we'll get the Lagrange interpolation:

$$L_n(x) = \sum_{j=1}^n y_j p_j(x) \quad (2)$$

As the operation above, the Lagrange Interpolation has been conducted and the theorem has been proved. In the later experiments, this interpolation is our important tool to do the fitting operation.

B. Similarity computation

After the above operation, we have got the fitting polynomial curve on the given several discrete points. Imagine that there are two trajectories, the target T and the compared one, S , represented by several latitude/longitude points individually. Using Lagrange Interpolation, we transform the trajectories into two fitting polynomial curves. And now, the problem transforms to how to measure the similarity of this two curves. In this paper, we are trying to apply a deformed integration to compute the similarity.

Conventionally, it is natural to integrate the difference between the two curves. Suppose that this two curves are $T(x)$ and $S(x)$. And the compared trajectory S is recorded from point a to point b ,

$$\int_{a \rightarrow b} |T(x) - S(x)| dx \quad (3)$$

Equation (3) roughly measures the similarity of the two trajectories. Intuitively, the more similar T and S are, the smaller (3) is. It is based on the start and end point of the compared trajectory. Logically, we want to find a trajectory similar to the target. As our experiments set, the compared trajectory is often shorter than the target one.

However, the experiments show that the results of the method above is not very acceptable and fails our original expectation at least. So we go about improving the integration.

There are some abuse in (3). It imports many meaningless features. As we operated before, we applied Lagrange Interpolation to transform the discrete points into the fitting curve. But when we get this step, we actually input countless points irrelevant to the trajectory into the 2-D Euclidean space. And once we adopt (3) as the similarity computation, we just add some meaningless number to the final result, which is proved to influence the result much. Not only in the accuracy, this integration also does not distinct the traditional trajectory similarity computation methods in the efficiency. We need to find the most similar trajectory for the target trajectory in a

label set. This integration requires to compute the similarity between the target and the undetermined trajectory dynamically as most the traditional method would do, which leads to time complexity $O(n^2)$. We need a more efficient method.

Our improvement is as follow. We just consider every single trajectory. And we reserve the Lagrange Interpolation to transform a trajectory T to a fitting curve $T(x)$. To avoid importing other meaningless features, get the span of this trajectory from the start point to the end point and divide it into m parts (1000 as our experiments set), calculate the value corresponding to the interval point on $T(x)$ and sum them all finally. Abstractly, we can regard the sum as the inherent "similarity property" of its corresponding trajectories. And the similarity measure of two trajectories is the absolute value of difference of the respective so-called similarity property, which leads to time complexity $O(n+v)$, where v represents the average length of the trajectories.

Suppose that there are a number of discrete points representing trajectory T in the interval $[p, q]$, and its similarity property is:

$$\sum_{i=0}^{m-1} T(p + \frac{q-p}{m}i) \quad (4)$$

For each trajectory, we can get (4) as the property is inherent. And measuring the similarity depends on the absolute value of the difference of (4). The algorithm is shown in Algorithm 1.

Algorithm 1:

Input: The original trajectory points sequence T ,
corresponding fitting curve $T(x)$,
interval m
similarity property P 's set D of the label set

Output: The rank of T 's true label

```

1 for  $u$  in  $m$  do
2    $P \leftarrow P + T(u)$  according to equation (4)
3 end for
4 sort  $D$ 
5 rank  $\leftarrow 0$ 
6 for  $i$  in  $D$  do
7   rank  $\leftarrow$  rank + 1
8   if  $P < D(i)$  do
9     break
10  end if
11 end for
12 return rank
```

C. Map trajectory to larger space

We get the complete measure of the trajectory similarity computation. But for the latitude/longitude space, the prepare is not enough for our experiments.

In order to pursue the accuracy and efficiency, we lose the stability of the results to some extent. Though such sacrifice is worthy and acceptable, we still want to debauch the loss and enhance the stability as much as possible.

And in this part, we propose an improved method which is to enlarge the space. For the latitude/longitude space, the points representing the trajectories are so intensive as 1 longitude difference is more than 100 km, that when we adopt the Lagrange Interpolation, the fitting curve would flog as the interpolation needs to cross all the discrete points. Flogging makes distortion. To make the fitting curve more smooth, we choose to enlarge the space.

The core rule is to reserve the relative distance between points but to enlarge the absolute distance. For simplicity, select a constant α and α times the distance between the points by fixing the start point of the trajectory. After such operation, we have enlarged the space.

And the experiments show that after enlarging the space, the result is more steady.

D. Noise and varying sampling rates

In fact, noise is common in the collected trajectory data. To simulate the actual situation, we need to add the noise to the collected trajectories. Firstly, we confirm a distorting rate r_l and randomly select a fraction of the points (indicated by r_l) and then, distort them by adding a Gaussian noise with a radius 30 (meters). Suppose that a point (x, y) is distorted,

$$\begin{aligned} x &= x + 30 \cdot dx, & dx &\sim \text{Gaussian}(0,1) \\ y &= y + 30 \cdot dy, & dy &\sim \text{Gaussian}(0,1) \end{aligned} \quad (5)$$

We now have the environment with noise. And another problem -- non-uniform sampling rate is left to cope with.

Due to the reality, such as the devices' inherent flaw and energy concern, we can not always obtain the regular lattice of trajectory points sequence. Likewise, we set the sampling rates.

Before we talk about the sampling rates, suppose that we have a points sequence \mathbf{T} , $\{T^{(i)}\}_{i=1}^n$, where n is the length of the points sequence \mathbf{T} , and we regard it the real route. Then, we create a sample from \mathbf{T} by randomly dropping some points with the sampling rates r_2 . For reality concern, we reserve the start point and the end point to avoid changing the underlying route of the sampled trajectory. In our experiments, the sampling rates varies from 0.2 to 0.6, and we compared with other traditional methods on the basis.

IV. EXPERIMENTS

We aim to verify the accuracy and efficiency of our proposed method on a taxi dataset. Compared to other three trajectory similarity computation methods which had been proved validly, we want to find our method outperforming in both accuracy and efficiency.

A. Experiment setup

Dataset: Our experiments are conducted on a real-world taxi dataset (<http://www.geolink.pt/ecmlpkdd2015-challenge>). It's collected in the city of Porto, Portugal. It contains over 1.7 million

trajectories. Each taxi reports its location at 15 second intervals. So each record consists of a series latitude/longitude points representing the trajectory. To reduce the contingency, we remove the trajectories which are less than 30 points. And there are about 1.2 million trajectories left.

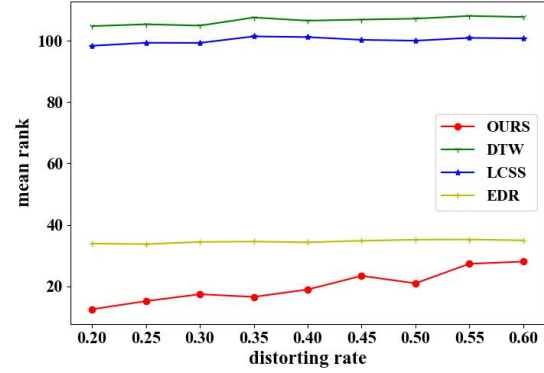


Fig. 2: Mean rank versus distorting rate

Benchmarking Methods: We compare our method with three other traditional methods in accuracy and efficiency, namely DTW[1], LCSS[2], EDR[4]. The above three methods are applied in trajectory similarity computation in most cases. To analyze the performance of our method and other methods in the round, we set many contrast experiments. We fix the sampling rate in 0.6, and vary the distorting rate in [0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60]. Then, we fix the distorting rate in 0.4 and vary the sampling rate in [0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60]. In every same parameter setting, we try to make our method outstanding.

LCSS and EDR are two of the most widely adopted trajectory measures in spatiotemporal data analyses. LCSS adopts the length of the longest common sub-sequence as the measure of the trajectory similarity computation, while EDR adopts the edit distance. To a certain extent, this two methods are based on the point-to-point match. They seek the as much as more matched points pairs. And obviously, they both abandon the regularity of the points sequence in the x axis, leading to the inaccuracy. And DTW measures the similarity by warping the sequence to best achieve “feature to feature” aligned. But in the trajectory similarity computation, the warping operation may distort the original features of the points sequence, which would also result in the inaccuracy.

Our method proposes a different thinking relative to above three traditional methods. A mixture of the interpolation and “deformed integration” helps to keep the original features and avoid bringing in too many redundant ones.

Evaluation Platform: Our method is implemented in Python running in Windows 10 with an Intel Core i7-7700 CPU.

B. Performance evaluation

The trajectory dataset we adopt in our experiments is consist of points sequences representing trajectories, which means that the dataset lacks the ground-truth that makes it hard to evaluate the performance of our method and other compared methods in accuracy and efficiency.

TABLE I: Four methods' mean rank versus sampling rate and distorting rate under 6:4 (a), 7:3 (b) and 8:2 (c) proportions of test set and label set

OURS LCSS DTW EDR	S. r a t e	0.2	0.4	0.6
		D.rate		
0.2		109.30	28.72	10.08
		69.70	64.28	61.75
		148.23	97.41	87.27
		62.98	46.40	23.80
0.4		112.01	30.65	10.93
		71.64	66.79	62.13
		149.96	100.35	88.46
		63.88	44.98	23.61
0.6		136.77	31.84	18.71
		71.90	66.62	59.67
		151.80	101.56	90.10
		62.48	45.19	24.25

OURS LCSS DTW EDR	S. r a t e	0.2	0.4	0.6
		D.rate		
0.2		77.20	18.73	9.89
		46.84	43.41	42.94
		122.56	76.84	66.31
		45.16	31.86	16.54
0.4		100.42	26.56	10.28
		47.16	45.60	43.69
		123.36	75.10	67.57
		45.78	31.65	16.38
0.6		98.96	30.25	11.70
		45.85	45.24	44.06
		125.23	78.59	68.69
		46.67	31.92	16.40

OURS LCSS DTW EDR	S. r a t e	0.2	0.4	0.6
		D.rate		
0.2		30.12	18.06	11.12
		33.09	31.63	30.18
		81.48	49.69	44.12
		31.68	20.65	18.15
0.4		43.92	19.58	15.41
		34.19	31.28	31.09
		85.68	51.92	44.98
		32.54	21.38	19.35
0.6		55.06	22.08	15.68
		33.94	32.37	32.01
		86.70	52.66	45.79
		32.43	24.87	20.63

(a)

OURS LCSS DTW EDR	S. r a t e			
		0.2	0.4	0.6
D.rate				
0.2		77.20	18.73	9.89
		46.84	43.41	42.94
		122.56	76.84	66.31
		45.16	31.86	16.54
0.4		100.42	26.56	10.28
		47.16	45.60	43.69
		123.36	75.10	67.57
		45.78	31.65	16.38
0.6		98.96	30.25	11.70
		45.85	45.24	44.06
		125.23	78.59	68.69
		46.67	31.92	16.40

(b)

OURS LCSS DTW EDR	S. r a t e			
		0.2	0.4	0.6
D.rate				
0.2		30.12	18.06	11.12
		33.09	31.63	30.18
		81.48	49.69	44.12
		31.68	20.65	18.15
0.4		43.92	19.58	15.41
		34.19	31.28	31.09
		85.68	51.92	44.98
		32.54	21.38	19.35
0.6		55.06	22.08	15.68
		33.94	32.37	32.01
		86.70	52.66	45.79
		32.43	24.87	20.63

(c)

We adopt a method called self-similarity[6] to perform the evaluation for our experiments. We randomly choose 10,000 trajectories from the dataset and for every sequence in the chosen trajectories, we down-sample and distort them to generate the sub-sequence due to the current sampling rate and distorting rate. And we get the label set of 10,000 trajectories.

And for every trajectory T from another 10,000 trajectories of the dataset, we can also get its sub-sequence S according to the current sampling rate and distorting rate. We call S the label of T . Now we get the point pair (T, S) and we regard T as the true route in reality, and S as the sampled data collected by the devices. Then we can say that S is the best match for T .

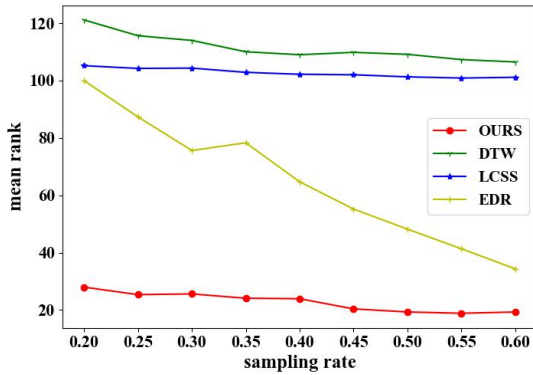


Fig. 3: Mean rank versus sampling rate

A good performance evaluation should keep stable regardless of the sampling or distorting strategy. And the self-similarity satisfies the requirement. So the next operation is to hide S in the label set. We try to find the k-nearest neighbors and figure out where the best match S ranks in the whole label

set. And we compare the mean-rank of our method to the other three ones.

C. Experiment in accuracy

As shown in Fig. 2, when the distorting rate is increasing, every method's mean rank augments generally, which corresponds to reality. The more distort, the more unmatched it gets. Likewise, in Fig. 3, as the sampling rate increases, the mean rank lowers by and large, which means the matching result is getting worse.

The above experiments show that the performance evaluation we adopt -- self-similarity is of stability and it is well adapted in our experiments proceeding. Self-similarity depicts the trend well as the distorting/sampling rate varies and the difference of all the compared methods under the same condition, which all render certain to the self-similarity for experiments.

And our experiments totally satisfy our previous thought. In every distorting rate and sampling rate, our method's mean rank is lower than other three ones, which means our method stands out. To achieve high accuracy, we pay the price of stability to some extent as the Fig shows. And we "enlarge the space" to minimize the cost as we originally wish to. It does work well in our experiments. The curve of our method increases/decreases steadily due to the "enlarge" operation, where we overcome the inherent defect to a large extent. The cost of the stability to enhance the accuracy is worthy.

The above experiments is conducted on basis of 5:5 proportion of the test set and label set. To explore further, we conduct experiments of 6:4, 7:3 and 8:2 proportions, seen in TABLE I, sampling rate denoted as S. rate and distorting rate denoted as D. rate. Due to interpolation, our method performs ordinarily while sampling rate gets low and the fitting curve distorts much, but stands out in most other situations.

D. Experiment in efficiency

Our method aims to stand out in efficiency, too. In the previous method analysis, our method involves “similarity property” in every trajectory, which reduces the time complexity. And we have recorded the running time of the compared methods, seen in Fig. 4.

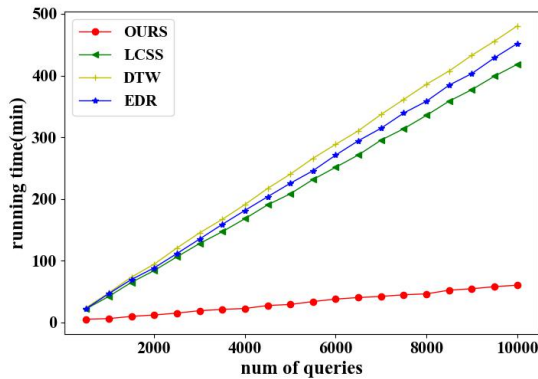


Fig. 4: Comparison of running time

We recorded the running time of our method compared to the traditional methods. And we can get that our method is outperforming in efficiency.

E. Time complexity

In this part, we discuss the time complexity of our method compared to other traditional trajectory similarity computation methods.

Our method does not dynamically compute the similarity of trajectories. Instead, we introduce a concept of similarity property. And for every single trajectory, we just compute its similarity property. Then, all the trajectories has their own similarity property. So we just need to compare the similarity property to each other and we get the final results. As we analyse before, it is evident that our method’s time complexity is $O(n+v)$, where v represents the average length of trajectories.

Our method decreases the time complexity by almost one order of magnitude. But as sacrifice, we lose a little stability to some extent which we view worthwhile.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a trajectory similarity computation method -- a mixture of Lagrange Interpolation and a kind of deformed integration and we compare our method to the traditional similarity computation methods in accuracy and efficiency.

And we set the contrast experiments, which then shows our method stands out in both accuracy and efficiency compared to other methods at the price of stability to some extent. Then, we minimize the sacrifice in stability by the operation of “enlarge” the longitude/latitude space, which also achieves great result.

To evaluate the performance of our method compared to others, we exploit self-similarity, which meets the need of evaluation in performance analysis, such as stability and intuition.

There are some problems remained. On the basis of keeping the satisfactory time complexity, we can explore other measures to compute the similarity property. And we can also change the mean-rank evaluation and the distribution of top-k trajectories.

FUNDING

This work is supported by the Young Scientists Fund of the National Natural Science Foundation of China (Grant No. 61802343), Zhejiang Provincial Natural Science Foundation of China (Grant No. LGF19F020019).

REFERENCE

- [1] B.-K. Yi, H. Jagadish, and C. Faloutsos, “Efficient retrieval of similar time sequences under time warping,” in *ICDE*, 1998, pp. 201 – 208.
- [2] M. Vlachos, G. Kollios, and D. Gunopulos, “Discovering similar multidimensional trajectories,” in *ICDE*, 2002, pp. 673 – 684.
- [3] L. Chen and R. Ng, “On the marriage of lp-norms and edit distance,” in *PVLDB*, 2004, pp. 792 – 803.
- [4] L. Chen, M. T. Oszu, and V. Oria, “Robust and fast similarity search for moving object trajectories,” in *SIGMOD*, 2005, pp. 491 – 502.
- [5] S. Ranu, P. Deepak, A. D. Telang, P. Deshpande, and S. Raghavan, “Indexing and matching trajectories under inconsistent sampling rates,” in *ICDE*, 2015, pp. 999 – 1010.
- [6] X. Li, K. Zhao, G. Cong, Jensen. C and W. Wei, “Deep Representation Learning for Trajectory Similarity Computation,” in *ICDE*, 2018.
- [7] H. Su, K. Zheng, H. Wang, J. Huang, and X. Zhou, “Calibrating trajectory data for similarity-based analysis,” in *SIGMOD*, 2013, pp. 833 – 844.
- [8] D. Williams and G. Hinton, “Learning representations by backpropagating errors,” *Nature*, vol. 323, no. 6088, pp. 533 – 538, 1986.
- [9] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, “Discovery of convoys in trajectory databases,” *PVLDB*, vol. 1, no. 1, pp. 1068 – 1080, 2008.
- [10] S. Ranu, P. Deepak, A. D. Telang, P. Deshpande, and S. Raghavan, “Indexing and matching trajectories under inconsistent sampling rates,” in *ICDE*, 2015, pp. 999–1010.
- [11] E. Frentzos, K. Gratsias, and Y. Theodoridis, “Index-based most similar trajectory search,” in *ICDE*, 2007, pp. 816–825.
- [12] S. Sankararaman, P. K. Agarwal, T. Mølhave, J. Pan, and A. P. Boedihardjo, “Model-driven matching and segmentation of trajectories,” in *SIGSPATIAL*, 2013, pp. 234–243.
- [13] H. Wang, H. Su Datab, K. Zheng, S. Sadiq, and X. Zhou, “An effectiveness study on trajectory similarity measures,” in *Aliaustranase Conference*, 2013, pp. 13–22.
- [14] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.
- [15] Z. Chen, H. T. Shen, and X. Zhou, “Discovering popular routes from trajectories,” in *ICDE*, 2011, pp. 900–911.
- [16] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *NIPS*, 2013, pp. 3111–3119.
- [17] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.
- [18] S. Sankararaman, P. K. Agarwal, T. Mølhave, J. Pan, and A. P. Boedihardjo, “Model-driven matching and segmentation of trajectories,” in *SIGSPATIAL*, 2013, pp. 234–243.
- [19] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, “H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.
- [20] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, “On using very large target vocabulary for neural machine translation,” *arXiv preprint arXiv:1412.2007*, 2014.

Clustering algorithms performance analysis applied to patent database

Cinthia M. Souza

*Institute of Mathematical Sciences and Informatics
Pontifical Catholic University of Minas Gerais
Belo Horizonte, Brazil
cinthia.mikaela@sga.pucminas.br*

Magali R. G. Meireles

*Institute of Mathematical Sciences and Informatics
Pontifical Catholic University of Minas Gerais
Belo Horizonte, Brazil
magali@pucminas.br*

Paulo E. M. Almeida

*Intelligent Systems Laboratory
Federal Center for Technological Education of Minas Gerais
Belo Horizonte, Brazil
pema@lsi.cefetmg.br*

Abstract—The granularity of large patent classification systems hampers the reclassification process in which patent categories are broken down into smaller ones, suggesting new categories. As these groups belong to a constricted domain of knowledge, keywords and subject descriptors tend to be similar and therefore insufficient to differentiate documents. In this context, the identification of common cited references can be useful to define semantic relationship among patents. This work compares citation analysis based results obtained by three clustering algorithms, SOM networks, K-Means and Multi-SOM. An empirical experiment was conducted using a patent database from the United States Patent and Trademark Office with all patents of four subgroups classified by the Cooperative Patent Classification system. Practical results evaluated by statistical inference techniques showed that SOM performs better than the other algorithms to cluster that database. This study can contribute with the reclassification process for a subgroup level of current patent classification systems, demonstrating how citation analysis can be an alternative attribute to the automatic clustering process.

Index Terms—Clustering algorithms, Computational intelligence, Knowledge representation, Patent database, Statistical inference

I. INTRODUCTION

A patent is a public concession, whereby the government, in exchange for full disclosure of an invention, grants the inventor the right to exclude others for a limited time from making, using or selling this invention [1]. Patents are organized into classification systems according to their technical application and structural characteristics to aid the patenting and retrieval processes.

With the growth of digital patent collections, the number of patents at all levels of classification systems has been increasing and some groups need to be dismantled in order to generate new groups and facilitate access to information. Considering the patents subgroups, which are subsets in an equal knowledge area and have a lot of similar words in their

abstracts, it is a challenge to identify common characteristics using words as attributes of the clustering process.

The main objective of this work is to evaluate the performance of some algorithms of patents clustering using citations as attributes. For this, three clustering algorithms will be used on a United States Patent and Trademark Office (USPTO) patent database. This work was divided into five sections. Section II presents a description of the implemented algorithms and some related works. Section III presents the database and proposed methodology, while Sections IV and V show the results and final considerations.

II. THEORETICAL BACKGROUND

SOM networks are maps of artificial neurons developed by Teuvo Kohonen in the 1980s. These structures, based on topological maps present in the cerebral cortex, are responsible for the execution of the grouping process. Each input neuron is connected to an output neuron by its respective association weight. This network uses unsupervised learning. From the instant the network identifies the regularity between the input data, it generates internal representations to encode the input characteristics and automatically create new groups. These networks have the capacity for self-organization and are more similar to neurobiological structures than supervised networks. Many of the experiments reported in the literature describe the use of SOM in grouping documents so to organize them as an alternative format for information retrieval [2].

The Multi-SOM algorithm is an extension of the SOM algorithm. This algorithm uses simultaneously several maps of SOM to cluster input patterns. The amount of simultaneous maps is defined by the user. For the initial map, data training is performed using the SOM algorithm. For the generation of the next maps, the algorithm realizes the superposition and the communication between the previous and the current maps. To carry out the transition from one map to another, it is necessary to define the new nodes. These nodes are defined by using the

mean square composition of the four neighbors, directly from the lower level [3], [4].

Performing this procedure preserves the original neighborhood at the higher levels and also the topographical properties of the maps [3], [4]. The Multi-SOM algorithm used in this work was implemented using an R library called “multisom” [5] available for R language. This algorithm not only performs grouping of data but also estimates the optimum number of clusters. At the end, it returns the best result obtained for the input data.

K-Means is an unsupervised data clustering algorithm. The main idea of this algorithm is to define k centroids, one for each cluster. After defining the centroids, the algorithm associates the centroid with the closest data. Then the centroids are recalculated and the previous steps are repeated for the new centroids until the centroids no longer move or the stop criterion is met [6].

Meireles, Cendón and Almeida [7] presented a comparison of document clustering process using keywords and citations. The experiments were performed using a test database with 200 articles from a restricted knowledge domain. The first experiment used the keywords of the articles in the test database as attributes for the clustering process. The second experiment was carried out using the citations of the articles as attributes. Both tests were performed using a SOM network. The experimental results showed that, in a domain of restricted knowledge with a great similarity between documents, the use of keywords was not very efficient. On the other hand, the use of citations can be considered an important alternative.

Lai and Wu [8] proposed an approach to create a new patent classification system, to assist patent managers in evaluating the basic patents for a specific industry. Li, Chen, Zhang and Li [9] considered the structure of patent citation networks for patent classification. They adopted a Kernel-based approach to capture content information and citation-related information in patents, and their proposal outperformed Kernel’s which did not use citation network structures. Liu and Shih [10] combined content-based, citation-based and metadata-based classification methods to develop a hybrid-classification approach using a modified K-Nearest Neighbor (KNN) algorithm.

III. PROPOSED APPROACH

The database used in the experiment was extracted from USPTO. The Cooperative Patent Classification (CPC) system, which is used by USPTO, classifies patents into sections, classes, subclasses, groups, and subgroups. Figure 1 illustrates the organization of this patent database and it was highlighted the low level hierarchies subgroups used in this work.

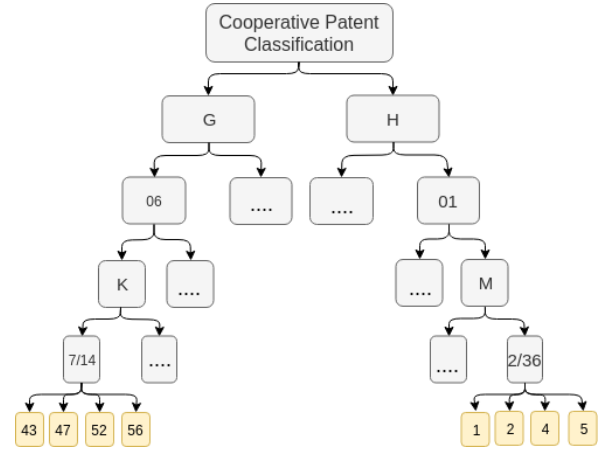


Fig. 1. Organization of the Database

In order to validate the proposed clustering process, two databases were created. Each database is composed by four subgroups randomly chosen in two distinct sections of the CPC system. In the hierarchy defined by CPC, the subgroups represent patent groups contextualized in the same area of knowledge. The first base is composed of subgroups G06K 7/1443, G06K 7/1447, G06K 7/1452 and G06K 7/1456 of the G06K subclass, named “recognition of data, presentation of data, record carriers, handling record carriers”, shown in Figure 1 only with their suffix 43, 47, 52 and 56. The second base is composed of subgroups H01M 2/361, H01M 2/362, H01M 2/364 and H01M 2/365 of the subclass H01M, named “processes or means, e.g. batteries, for the direct conversion of chemical energy, into electrical energy” represented in Figure 1 only with their suffix 1, 2, 4 and 5. Some patents of the subgroups are classified into more than one subgroup. In this work, only the first classification is considered and, therefore, the number of patents available is different from the number of patents selected for the database. Table I shows the name of the subgroups in the CPC system and the number of patents selected.

TABLE I
DATABASE

Sections	Codes CPC	Number of patents available	Number of patents selected
G	G06K 7/1443	505	452
	G06K 7/1447	302	263
	G06K 7/1452	93	78
	G06K 7/1456	227	117
H	H01M 2/361	213	185
	H01M 2/362	126	101
	H01M 2/364	33	28
	H01M 2/365	139	59

The methodology used to cluster the documents was divided into three execution steps and one analysis phase. In the first step, the citations of the patents contained in the selected subgroups were extracted. These citations were taken from the section of documents referenced by the patent. From this process, two binary citation matrices per document were

generated, one for each database, which inform the occurrence of a certain citation in each document. In these matrices, the digit 0 represents the absence of a citation in the document and the digit 1 represents the existence of a citation in the document. According to Borgman and Furner [11], the analysis of citations allows for the identification of relationships documents, regardless of the presence of equal terms. In this work, the occurrence of common quotes among patent documents is used as a mechanism to define the semantic relations between them.

In the second step, those matrices were used as inputs for each one of the three algorithms discussed in Section II. For each algorithm, the experiments were repeated for 30 times, to account for statistical validation. For the SOM network and K-Means, the number of clusters k was defined as 4, which was the number of subgroups of the database used for validation. The Multi-SOM does not need to receive as input the number of clusters. However, it is necessary to define the dimensions of the first map. Thus, the first map dimension was defined as 6×6 .

In the third step, an algorithm was implemented to evaluate the correspondence between the groups generated by SOM, Multi-SOM and K-Means algorithms and the original CPC subgroups presented in Table I. Finally, in the analysis phase, an objective comparison was performed using statistical inference, by hypothesis tests. In this test, the hypothesis tested (H_0) was the equality of average between the number of patents identified in step 3 and the number of patents selected in each subgroup. In this test, a categorization algorithm will be considered more efficient when the average is closer to zero. At the end, it was possible to infer if some of the algorithms were better or worse than the others, to solve that clustering problem. The alternative hypothesis (H_1) used for the Kruskal-Wallis H test and boxplot was the average difference in the number of patents clustered (ADPC) by each algorithm in the clusters, i.e. the difference between the number of patents identified, for example in group G1, and the number of patents selected in the corresponding original CPC subgroup. A low ADPC informs that the associated clustering algorithm groups together a number of patents similar to those of the original CPC subgroups.

IV. EXPERIMENTS

The experiments were divided into two phases. In the first phase, the tests were carried out with the database composed by the patents of section G. A total of 10,148 citations were extracted from the 910 patent documents. The matrices generated in this process were used for the clustering process. Fig. 2 shows the distribution of patents in the four groups generated by SOM algorithm, referred here as G_G1_S , G_G2_S , G_G3_S and G_G4_S . Kruskal-Wallis H test was performed to compare those samples from each of the groups generated, the test suggested that there were statistical differences between the three samples (with $p_0 = 5.31 \times 10^{-18}$ for the group G_G1 and with $p_0 = 3.08 \times 10^{-18}$ for the group G_G2). To find the difference of samples, a comparison was

performed by means of boxplot representations. This resulted in the plots of Fig. 3 and Fig. 4. It is possible to infer, with 95% of confidence, that SOM algorithm performs better than K-Means and Multi-SOM to cluster patents of this database by means of citations, as SOM averaged differences between number of patents identified on step 3 and patents selected in each subgroup (H_1) could not be rejected. Therefore, it is clear from this analysis that SOM performed much better than K-Means and Multi-SOM, as their ADPC is smaller than those by its counterparts.

To the groups identified by K-Means algorithm, it was added the termination KM (G_G1_KM , G_G2_KM) and to those identified by Multi-SOM, it was added the termination MS (G_G1_MS , G_G2_MS).

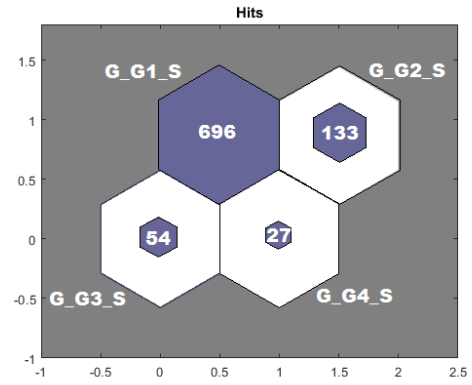


Fig. 2. Typical result of the clustering process using a SOM network

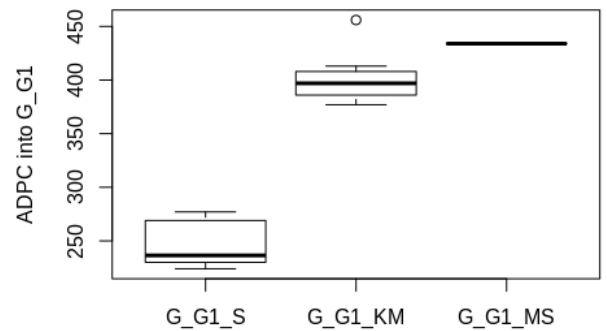


Fig. 3. Boxplot representation of results for G_G1

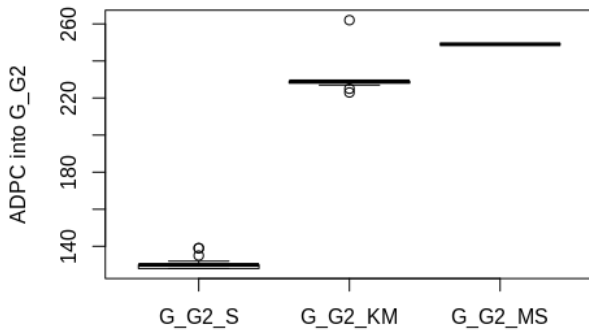


Fig. 4. Boxplot representation of results for G_G2

From the four groups created in the experiments, it was possible to identify two of them, which had a majority of patents, one from G06K 7/1443 and the other from the G06K 7/1447 subgroup. The other two groups had patents from the four used subgroups. Analyzing the first one created by SOM, named G_G1_S , with 696 averaged patents, 62.75% belonged to G06K 7/1443 subgroup of the documents database. K-Means and Multi-SOM created a big group with averages of 844.6 and 886 patents, respectively. From these, 58.60% and 54.85% in average were from the G06K 7/1443 subgroup. This indicates that K-Means and Multi-SOM algorithms failed to identify differences between documents from their citations, keeping the vast majority of patents in a single group.

The second group analyzed, created by SOM, named G_G2_S , had 133 patents in average. A total of 100% belonged to the G06K 7/1447 subgroup of the documents database. K-Means and Multi-SOM algorithms kept, in average, 100% and 28.57% of the patents from the G06K 7/1447 subgroup in G_G2_KM and G_G2_MS . But, while SOM was able to keep an average of 133 patents on G_G2_S , K-Means and Multi-SOM clustered only 35 patents in G_G2_KM and 14 patents in G_G2_MS respectively, in average. Table II presents the results obtained.

TABLE II
RESULT OF THE FIRST PHASE OF THE CLUSTERING PROCESS

Groups	Average cluster size	Average hit percentage
G_G1_S	696	62.75%
G_G2_S	133	100%
G_G1_KM	845	58.6%
G_G2_KM	35	100%
G_G1_MS	886	54.85%
G_G2_MS	14	28.57%

In the second phase, the tests were performed with the database composed of patents of section H. A total of 2,755 citations were extracted from the 373 patent documents. Fig. 5 shows the distribution of patents in the four groups generated by SOM algorithm, referred here as H_G1_S , H_G2_S , H_G3_S and H_G4_S . The Kruskal-Wallis H test suggested that there were statistical differences between the three samples of the three groups generated (with $p_0 = 1.26 \times 10^{-18}$ for the group H_G1 , $p_0 = 8.99 \times 10^{-19}$ for the group

H_G2 and $p_0 = 6.05 \times 10^{-17}$ for the group H_G3). To find the difference of samples, a comparison was performed by means of boxplot representations. The Fig. 6, 7 and 8 present a boxplot representation of the results in terms of ADPC. It is possible to infer, with 95% of confidence, that SOM algorithm performs better than K-Means and Multi-SOM to cluster patents of this database by means of citations, as SOM averaged differences between number of patents identified on step 3 and patents selected in each subgroup (H_1) could not be rejected. Again, it is clear that SOM's ADPC is significantly smaller than ADPC obtained by K-Means and Multi-SOM.

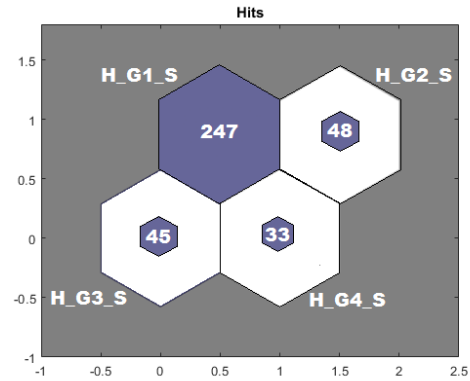


Fig. 5. Typical result of the clustering process using a SOM network

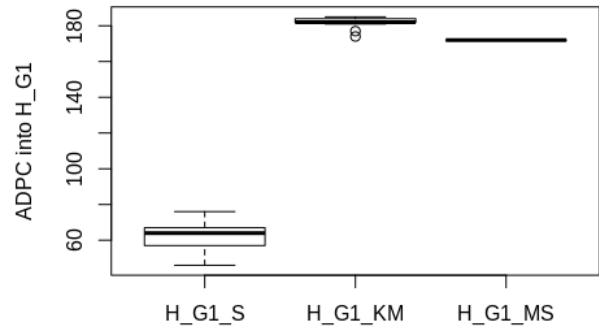


Fig. 6. Boxplot representation of results for H_G1

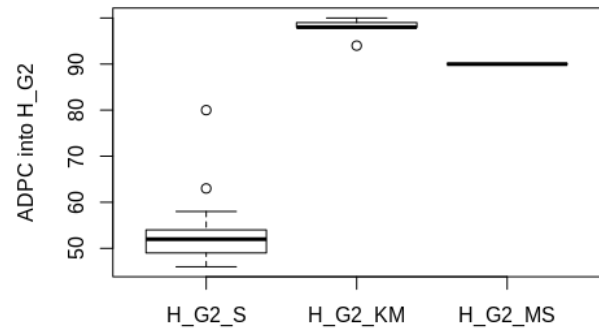


Fig. 7. Boxplot representation of results for H_G2

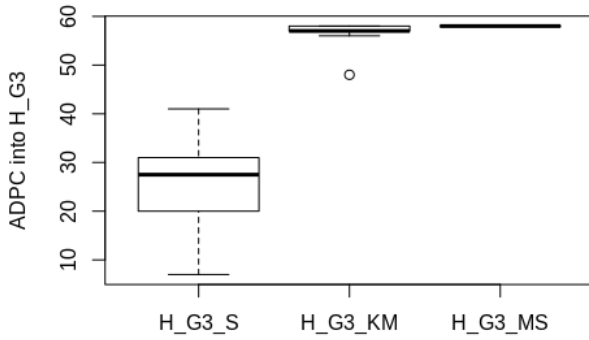


Fig. 8. Boxplot representation of results for H_G3

From the four groups created by the algorithms, it was possible to identify three with more similarity to the subgroups created by the specialists. These are equivalents to subgroups H01M 2/361, H01M 2/362 and H01M 2/365. These groups were named as H_G1 , H_G2 and H_G3 , respectively. The H_G1_S group created by SOM contains 247 patents in average, of which 57.82% were correctly grouped. K-Means and Multi-SOM have created the group H_G1_KM with 367 and H_G1_MS with 357 patents in average. From these, 57.17% and 56.58% were correctly grouped.

Analyzing the groups H_G2_S and H_G3_S it was identified that only SOM managed to create groups of a relevant size. The H_G2_S group created by SOM contains 48 patents in average, of which 78.30% were correctly grouped. The H_G3_S group created by SOM contains 45 patents in average, of which 57.41% were correctly grouped. Therefore, it is possible to state that the SOM has a more satisfactory result than K-Means and Multi-SOM, since it can better identify the differences between patent documents. K-Means and Multi-SOM clustered the vast majority of patents into a single group. These algorithms were not able, in these experiments, to identify differences between the documents that allowed them to be clustered in different groups. We believe that SOM could perform better than Multi-SOM to solve a given problem, even being nothing but a special case of Multi-SOM, because SOM can be more specific and specialized than Multi-SOM, thus being more precise. On the other hand, Multi-SOM is more general, and perhaps capable of dealing better with different instances of the problem. The percentage of patents correctly grouped by algorithms is very close, in some cases, this is due to the fact that the number of patents in the generated groups is very small. Table III shows the groups created, the average size of each cluster and the average percentage of patents correctly classified.

TABLE III
RESULT OF THE SECOND PHASE OF THE CLUSTERING PROCESS

Groups	Average cluster size	Average hit percentage
H_G1_S	247	57.82%
H_G2_S	48	78.30%
H_G3_S	45	57.41%
H_G1_KM	367	57.17%
H_G2_KM	3	58.70%
H_G3_KM	1	36.06%
H_G1_MS	357	56.58%
H_G2_MS	3	68.67%
H_G3_MS	2	50%

V. CONCLUSION

With the increasing number of patents and the development of new technologies, the classification systems employed by patent offices should be constantly reviewed to avoid accumulation of documents on certain subgroups. In a restricted domain of knowledge such as the subgroups of CPC system, it is difficult to use words as units of knowledge representation in an automatic clustering process because the subject descriptors and the words tend to be similar.

The main contribution of this work is to evaluate the performance of three clustering algorithms on a restricted knowledge domain, based on CPC sub-groups. The experiments brought the theory of citation analysis to a practical application of interest to the academic and industry communities. For the given scenarios, SOM networks showed superior performances compared with K-Means algorithm and Multi-SOM networks. Most of patent offices professionals and researchers in the domain of information retrieval and applied machine learning deal with the upper levels of classification hierarchies (class and subclass levels) and only some have tracked the problem on a more fine-grained classification (group and subgroup levels), as done in this work. For future work, it is expected to perform the comparison of the clustering process in a larger scale, using the upper hierarchy of the CPC system.

ACKNOWLEDGEMENTS

The authors would like to thank the financial support of the Pontifical Catholic University of Minas Gerais (PUC Minas), the Federal Center for Technological Education of Minas Gerais (CEFET-MG), the National Council for Scientific and Technological Development (CNPq, grant 429144/2016-4) and the Foundation for Research Support of the State of Minas Gerais (FAPEMIG, grant APQ 01454-17).

REFERENCES

- [1] T. Hufker and F. AlpertL, "Patents: a Managerial Perspective", Journal of Product and Brand Management, vol. 3, pp. 44-54, 1994.
- [2] X. Luo, A. N. Zincir-Heywood, "A comparison of som based document categorization systems", in: Proceedings of the International Joint Conference on Neural Networks, vol. 3, pp. 1786-1791, 2003.
- [3] X. Polanco, C. François, and J-C. Lamirel, "Using artificial neural networks for mapping of science and technology: A multi-self-organizing-maps approach", Scientometrics, vol. 51, pp. 267-292, 2001.
- [4] I. Khanchouch, M. Charrad, and M. Limam, "A Comparative Study of Multi-SOM Algorithms for Determining the Optimal Number of Clusters", International Journal of Future Computer and Communication, vol. 4, pp. 198-202, 2015.

- [5] S. Chair, M. Charrad, and N. Ghazzali, "A new r package for multi-som clustering", in *Conférences Conjointes Francophones sur la Sciences des Données AAFD & SFC*, 2016.
- [6] D. Xu, Y. Tian, "A comprehensive survey of clustering algorithms", *Annals of Data Science*, vol. 2, pp. 165-193, 2015.
- [7] M. R. G. Meireles, B. V. Cendón and P. E. M. Almeida, "Comparação do processo de categorização de documentos utilizando palavras-chave e citações em um domínio de conhecimento restrito", *Transinformação*, Campinas, vol. 28, pp. 87-96, 2016 (in portuguese).
- [8] K-K. Lai and S-J. WU, "Using the patent co-citation approach to establish a new patent classification system". *Information processing & management*, Elsevier, vol. 41, pp. 313-330, 2005.
- [9] X. Li, H. Chen and Z. Zhang and J. Li, "Automatic patent classification using citation network information: an experimental study in nanotechnology", in *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries (JCDL '07)*, pp. 419-427, 2007.
- [10] D-R. Liu and M-J. Shis, "Hybrid-patent classification based on patent-network analysis", *Journal of the Association for Information Science and Technology*, Wiley OnlineLibrary, vol. 62, pp. 246-256, 2011.
- [11] C. L. Borgman and J. Furner, "Scholarly communication and bibliometrics", *Annual review of information science and technology*, v. 36, n. 1, p. 2-72, 2002.

Automatic Calibration of Performance Indicators for Performance Analysis in Software Development

Mushtaq Raza

INESC TEC, Porto, Portugal/ Department of Computer Science
Abdul Wali Khan University Mardan
Mardan, Pakistan
mushtaq.raza@fe.up.pt

João Pascoal Faria

INESC TEC/ Faculty of Engineering, University of Porto
Rua Dr. Roberto Frias
Porto, Portugal
jpf@fe.up.pt

Abstract—ProcessPAIR is a novel method and tool for automating the performance analysis in software development. Based on performance models structured by process experts and calibrated from the performance data of many developers, it automatically identifies and ranks potential performance problems and root causes of individual developers. However, the current calibration method is not fully automatic, because, in the case of performance indicators that affect other indicators in a conflicting way, the process expert has to manually calibrate the optimal value in a way that balances those impacts. In this paper we propose a novel method to automate this step, taking advantage of training data sets. We demonstrate the feasibility of the method with an example related with the Code Review Rate indicator, with conflicting impacts on Productivity and Quality.

Index Terms—automatic performance analysis, personal software process, Performance analysis tool

I. INTRODUCTION

Process and product data produced in software development projects can be periodically analyzed to identify performance problems, determine their root causes and devise improvement actions. However, conducting the analysis manually is challenging because of the potentially large amount of data to analyze, the effort and expertise required, and the lack of benchmarks for comparison.

ProcessPAIR is a novel method and tool for automated performance analysis and improvement recommendation in software development [8]. Based on performance models defined by process experts and calibrated from the performance data of many projects, it automatically identifies and ranks potential performance problems and root causes of individual entities (developers, teams or organizations), so that subsequent manual analysis for the identification of deeper causes and improvement actions can be properly focused. ProcessPAIR was successfully applied in education and training environments [9]. ProcessPAIR is also of interest to high-maturity organizations (CMMI maturity levels 4 and 5), because it facilitates the implementation of practices of the Organizational Process Performance (ML4) and Organizational Performance Management (ML5) process areas [3].

However, the current calibration method used by ProcessPAIR is not fully automatic, because the optimal value of each

performance indicator must be provided by the process expert. In many cases, the optimal value follows directly from the definition and is located in one extreme of the scale (minimum or maximum). But in the case of performance indicators that affect other indicators in a conflicting way, an intermediate optimal value that balances those impacts need to be manually calibrated by the process expert. An example is the Code Review Rate (size unit reviewed per time unit). If code reviews are performed too fast, quality of reviews (Code Review Yield) is negatively affected, but if they are performed too slow, Productivity is negatively affected, and it is not easy to choose a review rate that balances these two conflicting impacts.

In this paper we propose a novel method to automate this step, and hence fully automate the model calibration process, taking advantage of training data sets. We show the feasibility of the method with a data set that includes Code Review Rate, Productivity and Code Review Yield data.

The article is organized as follows. Section II presents background information about ProcessPAIR. Related work is presented briefly in Section III. Section IV presents the proposed method and feasibility study. Section V concludes the article and points directions for future work.

II. BACKGROUND

A. The ProcessPAIR Approach

The ProcessPAIR approach involves three main steps (see Figure 1):

- 1) *Define*: Process experts define the structure of a performance model (PM) suited for the development process under consideration. In our approach, a PM comprises a set of top-level and child performance indicators (PIs), organized hierarchically by cause-effect relationships [7].
- 2) *Calibrate (or Learn)*: The PM is automatically calibrated by ProcessPAIR based on the performance data of many process users. The statistical distribution of each PI and statistical relations between PIs are computed from the training dataset, taking advantage of statistical and machine learning techniques [7].
- 3) *Analyze*: Once a PM is defined and calibrated, the performance data of individual entities can be automatically

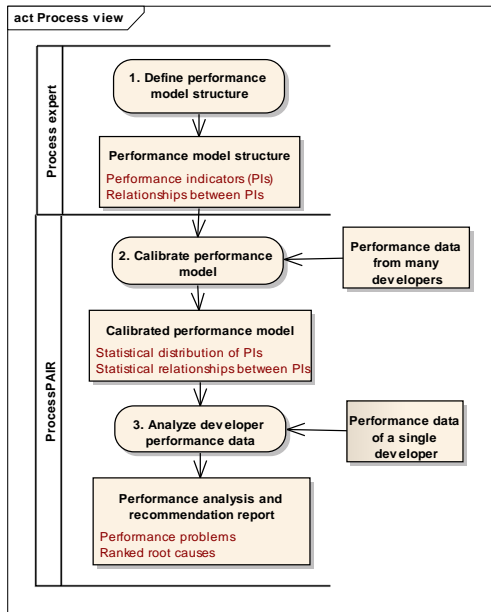


Fig. 1. UML activity diagram depicting the main activities and artifacts in the ProcessPAIR method.

analyzed by ProcessPAIR, to identify performance problems (in top-level PIs), identify potential root causes (related with child PIs), and rank those potential root causes.

The ProcessPAIR approach is supported by the ProcessPAIR tool, freely downloadable from <https://blogs.fe.up.pt/processpair/>. The tool is implemented as a standalone Java application, in order to protect the users data. It has a core framework and extensions for the processes of interest. An extension for the Personal Software Process (PSP), containing the definition of performance models for the PSP and data loaders from the most relevant project management tools used by PSP developers, was developed for education and training environments, but other extensions can be easily developed for other processes and contexts.

Further details about each step are given next.

B. Model Definition

The first step in our approach is the definition of the following elements of the PM:

- list of relevant PIs, including formulas for their computation from base measures, and the definition of the optimal value of each PI;
- subset of top-level PIs;
- cause-effect relationships between PIs, determined by a formula or statistical evidence;
- sensitivity coefficients [10] between PIs related by a formula (needed for ranking the identified root causes in the performance analysis step).

C. Model Calibration

The PM is automatically calibrated by ProcessPAIR from training data sets, generating the following data:

- approximate statistical distribution (cumulative distribution function) of each PI in the training data set;
- recommended performance ranges for each PI, needed for classifying values of each PI of a subject under analysis into three semaphores: green - no performance problem; yellow - a possible performance problem; red - a clear performance problem. Such ranges are calibrated automatically from the training data, so that there is an approximately even distribution of data points by the semaphores. In particular, the green range corresponds to the 1/3 data points closest to the optimal value, and the red range corresponds to the 1/3 data points farthest to the optimal value;
- regression models and sensitivity coefficients between PIs not related by a formula. Sensitivity coefficients between PIs not related by a formula are computed by first determining a regression model from the calibration dataset (a piecewise linear model organized as a regression tree [1]), and subsequently computing the corresponding sensitivity coefficient.

Some results of model calibration can be consulted in Figure 2. The example refers to the Code Review Rate, here named as Code Review Productivity. The approximate statistical distribution (cumulative distribution function) of this performance indicator, calibrated automatically by ProcessPAIR based on a training data set, is shown on the bottom left side. The 'green' and 'yellow' performance ranges are shown on the right; these ranges are calibrated automatically by ProcessPAIR, based on the commutative distribution function and the optimal value (calibrated manually by the process expert). The data points in the chart on the right show the values of this performance indicator for a series of projects under analysis. Different performance indicators defined in the performance model can be consulted in the tree view on the top left side. The Code Review Productivity has a green semaphore because its values lie mostly inside the green range.

D. Performance Analysis

Having defined and calibrated the PM, the performance data of individual entities (developers, teams or organizations) can be automatically analyzed by ProcessPAIR, to identify and rank performance problems and potential causes [7].

To rank the identified causes (child PIs) of performance problems in top-level PIs, it is used a ranking coefficient, that combines a sensitivity coefficient (measuring the impact of improving child PIs on top-level PIs) and a so-called percentile coefficient (measuring the difficulty of improving the child PIs).

The percentile coefficient is computed based on the distance of the observed values to the optimal value of each PI.

Hence the choice of optimal value has impact on both problem identification and root cause identification and prioritisation.

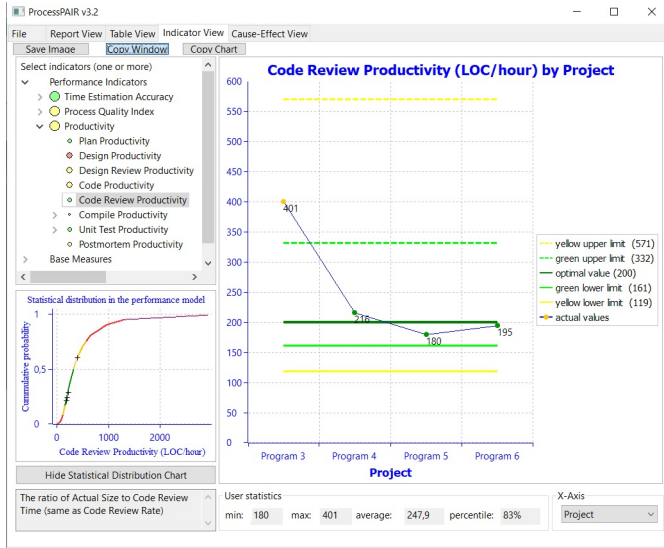


Fig. 2. ProcessPAIR user interface.

III. RELATED WORK

A. Optimal Code Review Rate

According to [4] [12], the time spent in reviewing a work product in relation to its size (review rate) is a leading indicator of the review yield (percentage of defects found).

In a published study [5], the recommended review rate of 200 lines of code (LOC) per hour or less was found to be an effective rate, identifying nearly two-thirds of the defects in design reviews and more than half in code reviews.

A team using the Team Software Process (TSP) obtained a process performance model (PPM) for establishing a target code review rate (number of lines of code reviewed per hour), based on the predicted impact on the code review yield (percentage of defects found in reviews), characterized as [12]:

- Regression equation: $CodeReviewYield = 146 - 0.364 \times CodeReviewRate$
- $R^2 = 94.1\%$, $p - value = 0.000$

According to this regression equation, the smaller the code review rate, the higher is the predicted code review yield (that, anyway, by definition, cannot exceed 100).

However, the quantitative impact on overall productivity was not analysed in those studies.

B. Productivity Measurement

Software development productivity is usually measured in function points per time unit or lines of code (LOC) per time unit [13] [6] [11]. However, both productivity measurement techniques have some limitations. On one hand, the measurement of function points remains subjective even after the completion of the software development project. On the other hand, productivity measures based on LOC have limitations due to the lack of counting standards and the dependence on the programming language [2].

In the data set we will explore for automatic calibration of the optimal value, there is no information about function

points, only size and time. The training data set contains data from more than 3000 individuals that developed the 10 projects of the standard PSP training (the same projects for all individuals, but with varying programming languages). Hence, we will take the average effort per project as a proxy for the functional complexity of each project, and calculate the individual productivity as the ratio between the functional complexity of the projects and the actual time (hours) spent by that individual.

IV. PROPOSED METHOD AND RESULTS

A. Method

Let us assume that a child PI X (such as the Code Review Rate) has conflicting impacts on two or more parent PIs Y_1, Y_2, \dots, Y_n (such as the Code Review Yield and Productivity).

The first step is to analyse the impact of the child PI X on a parent PI Y_i at a time, represented as a function f_i from X to Y_i . In order to arrive at a smooth function, we derive that function as follows: for each candidate optimal value x of X , we compute the mean value of Y in the data points that have the value of X within the green range corresponding to x .

Formally, denoting by S the training data set, p a data point in S , $Y_i(p)$ the value of Y_i in p , $X(p)$ the value of X in p , F the cumulative distribution function of X in S , and F^{-1} the inverse of F ,

$$f_i(x) \triangleq \text{mean}\{Y_i(p) | p \in S \cdot X(p) \in \text{Green}(x)\}$$

with

$$\text{Green}(x) = [F^{-1}(\frac{2}{3}F(x)), F^{-1}(\frac{2}{3}F(x) + \frac{1}{3})]$$

In the second step, we compute a combined impact function f_c , as a normalized average of the previous functions:

$$f_c(x) \triangleq \text{mean}\{\frac{f_i(x)}{\max(f_i)} | i = 1, \dots, n\}$$

The values of this function are adimensional values in the 0-1 scale.

Finally, we choose the value x of X that maximizes $f_c(x)$.

All the filtering procedures and calculations can be fully automated.

We implemented the calculations in a prototype tool taking advantage of evolutionary algorithms (genetic algorithms) to solve the optimization problem in a way that can scale to large data sets.

B. Results

In this study, for automatic calibration, we used a PSP data set available from the Software Engineering referring to 31,140 projects concluded by 3,114 engineers during 295 classes of the classic PSP for Engineers I/II training courses running between 1994 and 2005. In this training course, targeting professional developers, each engineer develops 10 small projects, following increasingly sophisticated process

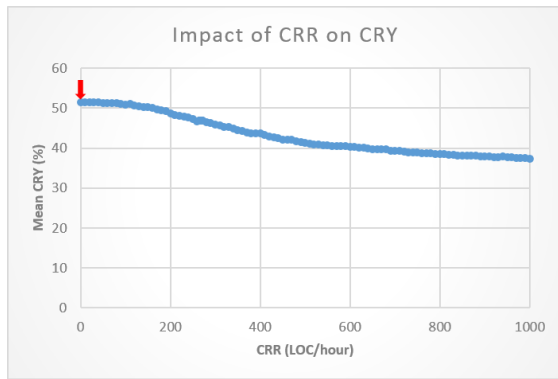


Fig. 3. Impact of CRR on CRY

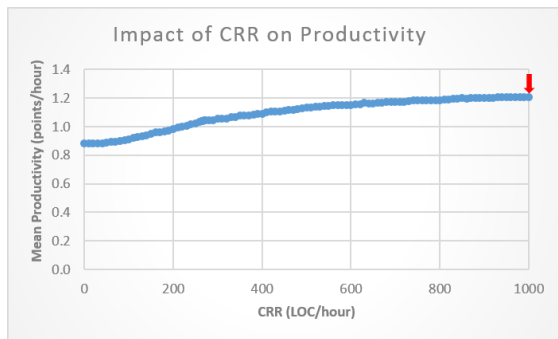


Fig. 4. Impact of CRR on Productivity

variants (PSP0, PSP1, etc.). Since code reviews are introduced only in the third project, we excluded the data points with zero time spent in Code Reviews. Since the Code Review Yield is undefined in case of 0 defects entering the Code Review phase, we also excluded data points with undefined Code Review Yield. In the end, we selected 9,650 data points (each corresponding to a project developed by a developer). Based on the selected data points, we computed the impact functions for the case of Code Review Rate, impacting Productivity and Code Review Yield. The resulting curves are presented in Figures 3, 4 and 5.

Figure 3 shows that, as expected, higher values of Code Review Rate are associated with lower values of Code Re-

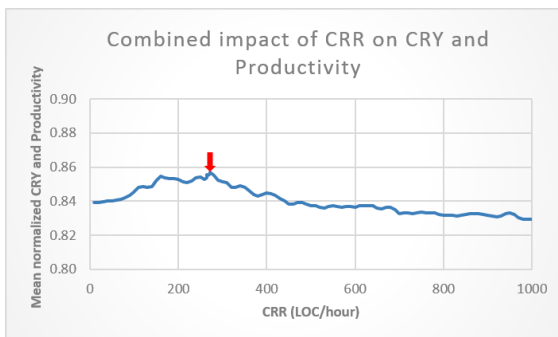


Fig. 5. Combined impact of CRR on CRY and Productivity

view Yield, which declines more significantly in the 200-500 LOC/hour range.

Figure 4 shows that, as expected, higher values of Code Review Rate are associated with higher values of Productivity, with a more significant increase in the 100-600 LOC/hour range.

Figure 5 shows the combined impact of Code Review Rate on Code Review Yield and Productivity. The combined curve has some oscillations due to the close symmetry of the two component curves, with a peak value at 270 LOC/hour.

Hence, the computed optimal value is 270 LOC/hour. This value is a bit higher than the literature recommendation of 200 LOC/hour, but perhaps closer to common practice when productivity impact is also important.

V. CONCLUSIONS

The method proposed in this paper worked successfully for the case study presented, allowing the automatic calibration of the optimal value and range of a PI (Code Review Rate) with conflicting impacts on other PIs (Code Review Yield and Productivity). The derived optimal value (270 LOC/hour) is a bit higher than the literature recommendation (200 LOC/hour), which is justified by the fact that we are quantitatively analyzing not only the impact on review effectiveness (yield), but also on productivity.

As future work, we intend to implement the calibration method in the ProcessPAIR tool in order to automatically calibrate all the PIs with conflicting impacts on high-level PIs.

REFERENCES

- [1] L. Breiman. *Classification and Regression Trees*. The Wadsworth statistics/probability series. Wadsworth International Group, 1984.
- [2] David N Card. The challenge of productivity measurement. In *Pacific Northwest Software Quality Conference*, pages 1–10, 2006.
- [3] Mary Beth Chrissis, Mike Konrad, and Sandra Shrum. *CMMI for development: guidelines for process integration and product improvement*. Pearson Education, 2011.
- [4] Watts S Humphrey. *Psp (sm): a self-improvement process for software engineers*. Addison-Wesley Professional, 2005.
- [5] Chris F Kemerer and Mark C Paulk. The impact of design and code reviews on software quality: An empirical study based on psp data. *IEEE transactions on software engineering*, 35(4):534–550, 2009.
- [6] Katrina D Maxwell and Pekka Forselius. Benchmarking software development productivity. *Ieee Software*, 17(1):80–88, 2000.
- [7] M. Raza and J. P. Faria. A model for analyzing performance problems and root causes in the personal software process. *J. Softw. Evol. Process*, 28(4):254–271, April 2016.
- [8] Mushtaq Raza and João Pascoal Faria. Processpair: A tool for automated performance analysis and improvement recommendation in software development. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016*, pages 798–803, New York, NY, USA, 2016. ACM.
- [9] Mushtaq Raza, João Pascoal Faria, and Rafael Salazar. Assisting software engineering students in analyzing their performance in software development. *Software Quality Journal*, pages 1–29, 2019.
- [10] Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global sensitivity analysis: the primer*. John Wiley & Sons, 2008.
- [11] Goparaju Purna Sudhakar, Ayesha Farooq, and Sanghamitra Patnaik. Measuring productivity of software development teams. 2012.
- [12] Shurei Tamura. Integrating cmmi and tsp/psp: using tsp data to create process performance models. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 2009.
- [13] Stefan Wagner and Melanie Ruhe. A systematic review of productivity factors in software development. *language*, 1989, 1980.

Knowledge Engineering Research Topic Mining Based on Co-word Analysis

Xiumin Liu

National Science Library
Chinese Academy of Sciences
Beijing, China
liuxiumin@mail.las.ac.cn

Zheng Liu

National Science Library
Chinese Academy of Sciences
Beijing, China
liuz@mail.las.ac.cn

Abstract—This paper collects research papers on knowledge engineering 2009-2018 from the internationally-authoritative citation database Web of Science and uses the co-word analysis method to analyze these contents of literature. By statistical analysis of keyword frequency, the paper compiles the co-word table of the high-frequency words, uses statistical analysis software to make hierarchical clustering, and summarizes hot research topics in the knowledge engineering area.

Keywords—knowledge engineering; co-word analysis; topic research; data process

I. INTRODUCTION

In 1977, Professor B.A. Feigenbaum, a computer scientist at Stanford University in the United States, presented a new concept of knowledge engineering at the 5th International Conference on Artificial Intelligence. He believes that "knowledge engineering is the principle and method of artificial intelligence, providing a means to solve problems that require expert knowledge to solve. The proper use of the knowledge acquisition, expression and reasoning process composition and interpretation is based on knowledge. An important technical issue of the system." This type of knowledge-based system is an expert system built through intelligent software.

The method of co-word analysis was first described in detail in the mid to late 1970s by French biometrician. Co-word analysis is a content analysis technique that uses patterns of co-occurrence of pair of items (i.e., words or noun phrases) in a corpus of texts to identify the relationships between ideas within the subject areas presented in these texts [1].

Based on the co-occurrence frequency of pairs of words or phrases, co-word analysis has proved to be a powerful tool for knowledge discovery in databases. Word clustering has also been profitably used in the automatic classification of documents, see [2]. The underlying assumption is that words that typically appear together should be associated with similar concepts.

II. DATA SOURCE

This paper uses the international authoritative citation database Web of Science published by the American Institute of Scientific Information (ISI) as the source of literature information. The paper intends to analyze the knowledge engineering research topics, so that the search criteria are "topic or search keyword is knowledge engineering", the search scope is 2009-2018, and the search time was February 12 2019 in the web of science database. The database has the added categories for each topic for more e. In the "knowledge engineering" subject, the "computer science interdisciplinary applications" or "computer science artificial intelligence" categories have been chosen for the limitation of the papers' content. A total of 1550 related articles were retrieved. Each article record contains multiple fields, such as title, author keywords, citations, publication date, etc.

III. RESEARCH METHOD

This paper uses the methods of bibliometrics and content analysis to quantitatively and qualitatively analyze the data. The Bibliometric Method is a quantitative analysis method that uses mathematical statistics to study the external features of the literature. Content Analysis Method is an in-depth analysis of the content of the research object [3].

There are two types of keywords in the research papers. One is author keyword (DE field), the other is a supplementary keyword (ID field) ISI keyword which is extracted by ISI according to the title of the paper. In order to fully reflect the topic of research in the field of knowledge engineering in the past ten years, using the author keywords.

After the DE field is split with punctuation ":", the list of keywords is formed. In the former similar papers directly used the form co-occurrence to establish a co-word matrix, this article will further summarize the word forms differently. Lemmatization rules apply to the keywords list. Lemmatization is a process of assigning a to each word form in a corpus using an automatic tool called a lemmatizer [4]. Lemmatization reduces inflected forms of a word to their lexical root. With lemmatization turned on, a keyword is reduced to its "lemma". As a result, lemmatization can reduce or eliminate the variant spellings of a word express the same

concept. For example: citing、cites、cited and citation are the inflected forms of the cite. Lemmatization makes the mapping between cite and its each inflected form. In this paper, stemming rules also apply to the keywords processing. With stemming, words are reduced to their word stems. A word stem need not be the same root as a dictionary-based morphological root, it just is an equal to or smaller form of the word [5]. Stemming removes suffixes such as -ing and -es from keyword in order to cluster relevant keywords. For example, with stemming rules the keywords “vinyl recording” will be same as vinyl record. Stemming and lemmatization are closely related. The difference is that stemming merely drops suffixes such as -ing and -es, while lemmatization makes use of dictionaries that define pairs and clusters (e.g., defense, defenses) of words with the same meaning or with a shared morphological structure. Both lemmatization and stemming apply only to English language search terms. After the prototyping process, calculate the keyword frequency. Get the lemma keywords list with frequency.

From the literature of 1550 articles, the author keywords were merged through prototyping and stemming, and more than 4,000 concepts were obtained. On the basis of high-frequency word statistics, a co-occurrence matrix as TABLE I is established according to the frequency selection high-frequency keywords which frequency is greater than or equal to 10.

In the construction of the author's key common word matrix, this paper uses the Python program to prototype and stem the author keywords to establish a 203*203 matrix. The matrix is a symmetric matrix in which the data on the main diagonal is defined as the conceptual frequency, and the data on the non-main diagonal represents the number of times the two concepts appear together in the same paper. The concept here can be composed of multiple forms.

IV. BIBLIOMETRIC ANALYSIS

A. Published Time Distribution

The change in the number of academic papers is an important indicator for measuring the development trend of a field for a period of time. It is of great significance for evaluating the stage of the field and the development trend and forecasting future trends. The author makes diachronic systematic statistics on the distribution of 1550 documents, and

makes a distribution curve scatter plot, as shown in Fig. 1.

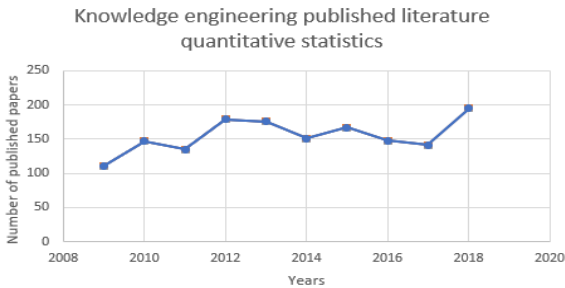


Figure 1. knowledge engineering published literature quantitative

It can be seen from the figure that the number of publications in knowledge engineering research has been growing steadily in the past decade, especially in 2012 and 2018, respectively, with two small peaks, 179 and 195. It shows that the industry has always attached great importance to the research of knowledge engineering, and knowledge engineering is still the current research hotspot.

B. Publications

Through statistical analysis, it is concluded that the top 24 journals published the most published papers accounted for 50% of the total papers in the entire knowledge engineering field. The journals of papers amount distribution as Fig.2. Providing high-quality academic papers for researchers in the

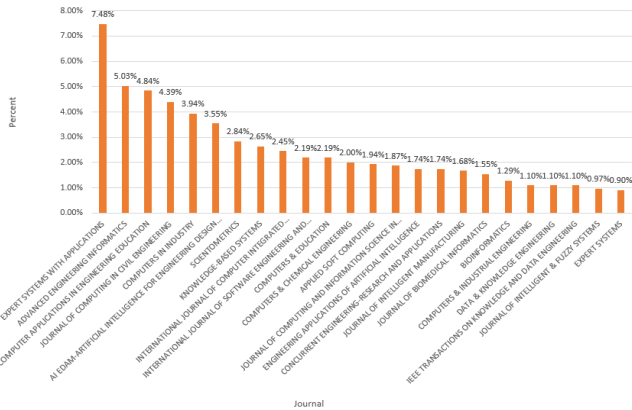


Figure 2. journal distribution

TABLE I. PART OF KEYWORD CORRELATION MATRIX

	Ontology	Knowledge engineering	Knowledge management	Ontology engineering	Knowledge representation	Knowledge-based engineering
Ontology	124	16	10	6	12	1
Knowledge engineering	16	88	3	1	10	1
Knowledge management	10	3	51	5	2	3
Ontology engineering	6	1	5	45	5	1
Knowledge representation	12	10	2	5	43	1
Knowledge-based engineering	1	1	3	1	1	30

field of knowledge engineering, researchers would publish their fresh and novel related papers or find fresh topic content.

C. Analysis of highly cited papers

In descending order of the order in which the documents are cited, select top10 as shown in the following TABLE II. According to the content of the article, the authors try to analysis the distribution of research topics in the field.

As can be seen from TABLE II, the most cited is Natural Language Processing (Almost) from Scratch [6] written by Professor Collobert, R *et al*, cited as 1010 times. This paper is one of the most classic papers of neural network language model and word embedding. Its core goal is to train good word embedding to complete tasks such as part-of-speech tagging, phrase recognition, named entity recognition and semantic role tagging. This paper provides a good idea for the future use of neural networks for specific natural language processing tasks. NLP technology can be used for text mining, analysis of natural language texts, discovery of conceptual relationships between knowledge points and knowledge points, and assisting in knowledge acquisition for large-scale knowledge engineering. Use NLP technology as a tool to practice knowledge engineering.

In August 1977, Edward Feigenbaum proposed the concept of knowledge engineering for the first time. Since then, knowledge engineering has been a foundation of expert systems (also known as knowledge-based system) [7]. Since the beginning of the 21th century, the knowledge engineering's evolution has been focused on big data. A vast amount of data is generated and processed every day, and accessible content on the Internet is far beyond the exploration capabilities of data consumers, who usually can't locate the relevant information within an acceptable time frame. To cope with the challenges brought by the big data phrase of knowledge engineering's evolution, BigKE (Knowledge engineering with Big Data) [8] uses its frame work to offer several advantages over conventional knowledge engineering.

V. ANALYSIS OF MAINSTREAM RESEARCH TOPIC

Knowledge engineering is an applied science with its rich research fields. Topic analysis is a method of content analysis that understands the internal state of development of a discipline. The author intends to use high-frequency keywords to co-occurrence matrix and analyze several mainstream subject areas of knowledge engineering research, and to study knowledge engineering from a vertical direction.

TABLE II. MOST CITED PAPER TOP TEN

Author name	Title	Publication Name	Cited_Times	Year_Published
Collobert, R; Weston, J; et al.	Natural Language Processing (Almost) from Scratch	JOURNAL OF MACHINE LEARNING RESEARCH	1010	2011
Wu, XD; Zhu, XQ; et al.	Data Mining with Big Data	IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING	700	2014
Margaryan, A; Littlejohn, A; Vojt, G	Are digital natives a myth or reality? University students' use of digital technologies	COMPUTERS & EDUCATION	256	2011
Hwang, GJ; Chang, HF	A formative assessment-based mobile learning approach to improving the learning attitudes and achievements of students	COMPUTERS & EDUCATION	235	2011
Rendle, S	Factorization Machines with libFM	ACM TRANSACTIONS ON INTELLIGENT SYSTEMS AND TECHNOLOGY	210	2012
Demirkan, H; Delen, D	Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud	DECISION SUPPORT SYSTEMS	181	2013
Barnaghi, P; Wang, W; et al.	Semantics for the Internet of Things: Early Progress and Back to the Future	INTERNATIONAL JOURNAL ON SEMANTIC WEB AND INFORMATION SYSTEMS	164	2012
Chu, HC; Hwang, GJ; Tsai, CC	A knowledge engineering approach to developing mindtools for context-aware ubiquitous learning	COMPUTERS & EDUCATION	154	2010
Li, JZ; Tang, J; Li, Y; Luo, Q	RiMOM: A Dynamic Multistrategy Ontology Alignment Framework	IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING	138	2009
Lu, JW; Liong, VE; et al.	Learning Compact Binary Face Descriptor for Face Recognition	IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE	135	2015

TABLE III. PART OF KEYWORD CORRELATION MATRIX

	Ontology	Knowledge engineering	Knowledge management	Ontology engineering	Knowledge representation	Knowledge-based engineering
Ontology	1	0.15316792	0.125749	0.080322	0.164337	0.016396
Knowledge engineering	0.153168	1	0.044781	0.015891	0.162564	0.019462
Knowledge management	0.125749	0.04478111	1	0.104371	0.042708	0.076696
Ontology engineering	0.080322	0.01589104	0.104371	1	0.113666	0.027217
Knowledge representation	0.164337	0.16256402	0.042708	0.113666	1	0.027842
Knowledge-based engineering	0.016396	0.01946247	0.076696	0.027217	0.027842	1
Expert system	0.016396	0.11677484	0	0	0.055685	0
Machine learning	0.032791	0.07784989	0	0	0.027842	0
Knowledge acquisition	0.016676	0.09897595	0.078008	0.083045	0.056637	0.067806
Knowledge-based systems	0.050913	0.12087344	0.026463	0.028172	0.057639	0.034503

Since the frequency of occurrence of each keyword itself is different, in order to eliminate the influence of the difference on the analysis below, the co-occurrence matrix is converted into the correlation matrix by Ochia coefficients, as show in TABLE III (part).The formula for calculating the Ochia coefficient is as follows: the Ochia coefficient of keywords K1 and K2 equals the number of co-occurrence of K1 and K2 is divided by A, and A is equal to the square of the product of the frequency of K1 and K2. According to the formula, the transformation of the co-occurrence matrix and the correlation matrix conduct.

The data in the correlation matrix is similar data, and its numerical value indicates the distance and similarity between the two keywords, that is, the larger the value, the closer the distance between the two keywords, the better the similarity [10]. In order to reduce the larger error caused by too many 0 values in the correlation matrix, the matrix is further transformed into a similar matrix, the larger the value, the closer the two keywords.

On the basis of the correlation matrix, get the hierarchical clustering. Euclidean Square Distance have been selected the measured range the in the process cluster analysis. The Average Linkage have been used for the distance of the clusters through the agglomerative hierarchical clustering technique. The result as Fig. 3 as below. According the research, the list of keywords frequency and the semantic relations between the keywords, the cluster tree divided into four sections: Smart learning and education, Knowledge acquisition, Knowledge basic algorithms and Knowledge engineering technology application.

A. Smart Learning and Education

Knowledge grid realizes the orderly organization and knowledge discovery of domain knowledge, which is the ideal environment for learning. The knowledge grid adopts ontology modeling based on description logic, which can realize reasoning and has great application prospects for satisfying individualized learning and guiding learning process [11]. The current far-reaching application of information technology in the field of education is about the study of educational

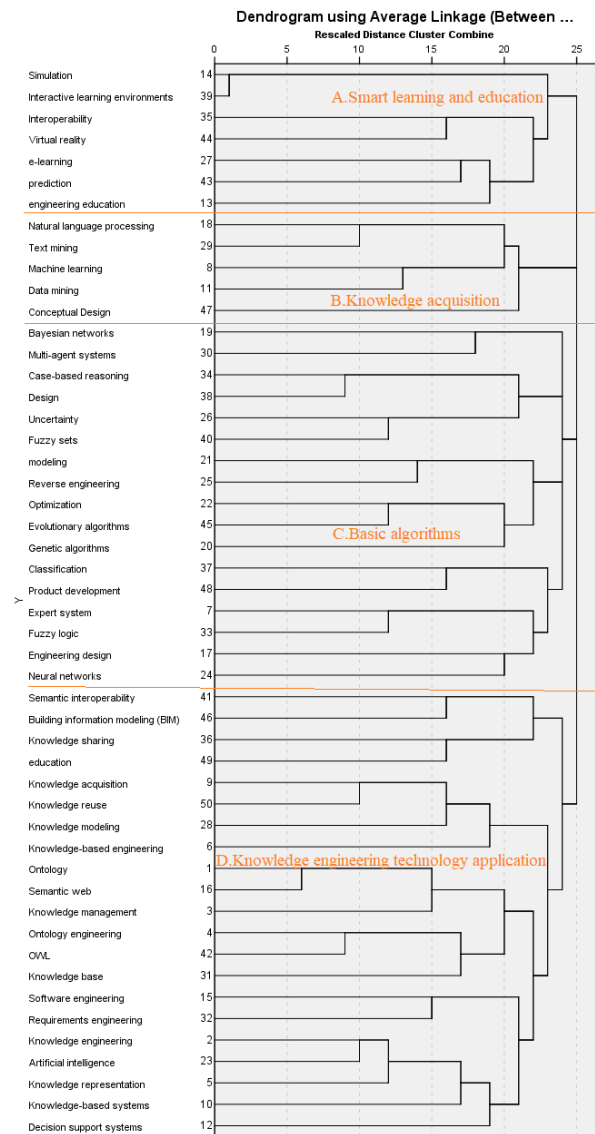


Figure 3. Keywords Hierarchical cluster tree

metadata, such as the Dublin Education Metadata Standard DC-Education and the IEEE-LOM Learning Object Model.

B. The Key Techniques of Knowledge Acquisition

"Technology" is the foundation for the application and innovation of knowledge engineering to realize its knowledge. This means. In the modern era of rapid development of modern information technology, key techniques of knowledge engineering are constantly innovating and advancing. "Knowledge Acquisition (KA)" is the process of extracting various expertise for problem solving from knowledge sources and transforming it into executable code on a computer [12]. Emerging knowledge acquisition techniques have Data mining, Web mining, text mining, etc., these techniques will help knowledge extraction in knowledge engineering and promote knowledge engineering to a deeper level.

C. Basic knowledge algorithms

Any calculation that follows the laws of nature can be called Computational Intelligence, sometimes called Soft Computing. It mainly includes three contents: fuzzy computing, which is introduced by human processing, fuzzy computing, which is based on the working rules of biological neural networks, and genetic algorithms and evolutionary calculations that mimic the "survival of the fittest" rule in the biological world (Evolution computing).

Machine learning processes can be effectively integrated into knowledge engineering pipelines using commonly available software frameworks that incorporate the mathematics and algorithms needed to perform deeper analysis than was possible before. Machine learning approaches fall into two broad classes: supervised learning and unsupervised learning. The supervised algorithms are given labelled training data, whereas unsupervised learning algorithms find structure within the input data. The construction of a knowledge engineering pipeline will typically need to leverage algorithms from both classes.

D. Knowledge Engineering Technology Application

Data science and technology have received widespread attention. Data-centric research methods and techniques in information, energy, medicine, sociology, etc. Various subject areas have been widely applied and recognized. Driven by the wave of urban informatization and the rise of data science, smart cities have become the new ideas and new practices of the next generation of urbanization on a global scale. "Smart City is built on the basic framework of digital cities. It is connected to the real city through the ubiquitous sensor network. The massive data storage, calculation, analysis and decision-making are handled by the cloud computing platform, and the results of the analysis decision are Automated control of various facilities" [13]. The information system as a smart city must have powerful computing power, sensing ability and data application capability, which will become the research field of knowledge engineering.

VI. CONCLUSION

With the help of the co-word analysis method, this paper analyzes the research topic in the field of knowledge engineering more intuitively and scientifically, and has made in-depth discussion, which has certain practical significance. Through the analysis of the authors, the following conclusions were obtained.

Starting from the analysis of the research topic based on co-word method. The word distribution of each document is used to establish a co-word matrix with lemmatization. After transforming it into a correlation matrix, it is hierarchically clustered by SPSS, and finally four subject groups are selected. In-depth content analysis, sorted out their main research content and hotspots.

ACKNOWLEDGMENT

This article has been helped, reviewed, and guided by Dr. Zheng Liu. I would like to express my heartfelt gratitude and sincere respect to her. The perseverance of the spirit of Dr. Liu, the profound knowledge, the work attitude of the facts, the generosity and the spirit of selfless dedication have always inspired me to constantly learn knowledge and pursue the realm of life. Thanks Dr. Haibo Li for his suggestion to modify the content of the article. There are too many people who have helped me, sincerely thank all those who care and help me, thank them for giving me the power to overcome myself, transcend myself, and constantly pursue, thank you all.

REFERENCES

- [1] Qin H. Knowledge Discovery Through co-word Analysis[J]. Library Trends, 1999, 48(1): pp. 133-159.
- [2] L. D. Baker and A. McCallum. Distributional clustering of words for text classification. In ACM SIGIR, 1998, pp. 96-103.
- [3] Wang Yuefen. Comprehensive study of bibliometrics and content analysis[D]. Wuhan: Wuhan University, 2007.
- [4] https://www.sketchengine.eu/my_keywords/lemmatization/.
- [5] <https://towardsdatascience.com/stemming-lemmatization-what-ba782b7c0bd8>.
- [6] Collobert R, Weston J, Bottou L, et al. Natural Language Processing (Almost) from Scratch[J]. Journal of Machine Learning Research, 2011.
- [7] R. Studer, V.R. Benjamins, and D. Fensel, "Knowledge Engineering: Principles and Methods," Data & Knowledge Eng., vol. 25, no. 1, 1998, pp. 161-197.
- [8] Wu X D, Chen H H, Wu G Q, et al. Knowledge engineering with big data[J]. IEEE Intelligent Systems, 2015, 30(5): 46-55.
- [9] Xiao Zhixiong, Gu Jing. Analysis of hotspots in domestic collaborative research based on co-word analysis [J]. Intelligence Exploration, 2015(5): pp. 6-14.
- [10] Key Technologies of Knowledge Measurement, Reasoning and Fusion in Knowledge Engineering Research [D]. Shanghai: Fudan University, 2004.
- [11] <https://towardsdatascience.com/stemming-lemmatization-what-ba782b7c0bd8>.
- [12] Ma Chuangxin. On Knowledge Representation [J]. Modern Intelligence, 2014, 34(3): pp. 21-28.
- [13] Li Deren, Shan Jie, Shao Zhenfeng, et al. Geomatics for Smart Cities-Concept, Key Techniques, and Applications[J]. Geospatial Information Science, 2013, 16(3): pp. 13-24.

Finding Erroneous Components from Change Coupled Relations at Fix-inducing Changes

Ali Zafar Sadiq, Ahmedul Kabir and Kazi Sakib

Institute of Information Technology, University of Dhaka

Email: zafarsadiq120@gmail.com, kabir@iit.du.ac.bd, sakib@iit.du.ac.bd

Abstract—During the gradual process of software evolution, errors appear in different components of a software system. These errors are later on fixed by developers as part of corrective maintenance activities. However, if errors appear continuously from a particular component, that may indicate design flaws or code smells. Maintenance cost will greatly reduce if design flaws are treated as early as possible. To find out such flaws it may require time-consuming manual inspections. This paper tries to find out such components using the information of change coupled cluster of files or Java classes at fix-inducing changes. In this proposed approach, information (like class, method, parameter of method and variable names) from change coupled relation of a class at Fix-Inducing Changes (FICs) are used to provide information about erroneous components. Then the error history, of software components, is found by using cosine similarity of information from change coupled cluster of classes found in FICs to see with the architectural information found from authenticated sources. Finally, the error history of components is shown as the percentage of change coupled cluster of a class found in FICs of each 100 commits in the version control system.

Index Terms—Fix-Inducing Change, Software Quality Assurance, Software Change, Software Maintenance, Change Coupling

I. INTRODUCTION

Change is an inevitable part of the evolution of software. Frequently co-changing software artifacts form change coupled relation. Any change in an artifact will influence change in other artifacts which are change coupled with the former. This relation can also be considered to form a cluster of artifacts with respect to a file or class, which may be affected depending on the change of that class. So any class and its co-changing artifacts can be considered to be a part of a module or component which shows close interactions among themselves.

Changes are done to introduce new features into the system or to fix existing errors and any changes can introduce errors, flaws or failures in the system. Various works identified these erroneous changes [1] and analyze their impact [2]. The reasons behind these changes may be improper coding or careless implementation of algorithms. For analyzing these changes, various properties of change like files affected, time, experience of developer and many others taken into consideration [3] [4] [5] [6]. However, none of those explored the architectural components of a software system is affected by those erroneous changes during the process of software development and maintenance.

Continuous appearance of errors from a particular component indicates that either that part has design flaws or it needs

redesigning. To find out such components, manual inspection of files from source repository and bugs from bug repository is required. Moreover, the bug repositories will only provide information about reported bugs whereas many unreported bugs fixed by developers will remain hidden. So considering bug repository may give less information than the actual situation.

Various works tried analyze the quality of software systems and condition of architectural components [7] [8] [9]. Evolution radar used change coupling relation to show the condition of software components [7]. This work did not consider the erroneous changes and only focuses on design flaws based on change couple relation. Using the comments from the version control system, sticky notes are seen to provide useful information [8] but the concept of error is not seen there. Furthermore, the relationship between the evolution of software artifacts and the way they are affected by problems is visualized by D'Ambros et al but it did not consider component based analysis using commit history [9]. To the best of author's knowledge, none of the existing works explored by combining the information of change coupling relation and Fix-Inducing Changes of source repositories to find software components error history.

To find the erroneous components, firstly the fix-inducing changes are found by tracking the modified and deleted of error fixing changes. Then the entire history is traversed using a commit window of 100 commits. In every 100 commits, the classes (only Java classes are considered in this work) found in the FICs are noted. Then for each of those classes, that class itself and the cluster of other classes forming change coupled relation with that class are considered. Then for each of these cluster classes, information about class, method, method's parameter, and variable names are collected. Using the cosine similarity of the information obtained from each cluster with the architectural information from the authenticated source, the most probable component for each cluster is found. Then of the total clusters, what percentage of clusters belongs to which component is graphically represented. So, the main contribution of this paper is to propose a methodology to generate the error history of software components for the entire lifetime of any source repository.

II. METHODOLOGY

This work tried to utilize the change coupling information of classes found in fix-inducing change. Unlike [7] [8] [9],

this work focuses on components error history. Analyzing this history might play an important role in the fields of software architecture, evolution, decay, and similar others.

The entire process of the proposed methodology consists of three steps. Firstly, fix-inducing changes are extracted by using historical information. Then clusters of change coupled classes are identified by observing their changing relation. Lastly, the error history is analyzed by using cosine similarity between obtained change coupled cluster of classes and architectural information.

A. Finding Fix-Inducing Changes (FIC):

Bugs are errors of the system that causes the system to behave in unintended ways. The origin of the bugs are FICs which introduce errors in the software system. These are also known as bug introducing change, Figure 1 explains the entire process of finding fix-inducing changes. This process starts with finding fixing commits which are mainly committed in the version control system by developers with a comment containing keywords like “Fix”, “Bug”, “Patch” or their past and gerund form. Any number with hashtags indicating bug number along with those keywords were also considered to represent fixing commits. These commits contain the change to correct errors. This change is done by modifying or deleting some lines of code that is present in the immediate commit parent to a fixing commit. To obtain those lines that were modified or deleted, Diffj tool’s [10] source code is used after modifying it according to our need. Since Diffj ignores white space and other format changes, so it ignores the possibility of finding false FICs as mentioned by [11].

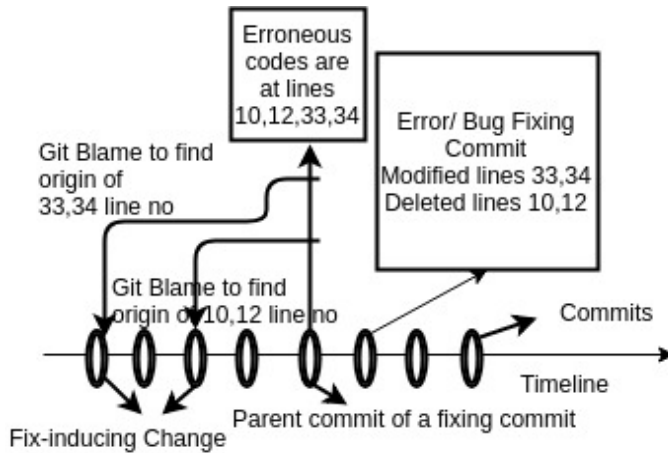


Figure 1: Finding Fix-Inducing Changes.

To track the origin of those lines in the parent commit of fixing change, Git blame command¹ is used [12]. The commits which were found are the FICs or changes introducing errors in the system. All FICs of the repository are found in this way.

¹git -c core.abbrev=40 blame -L(line number),+1 (FCParentHash) ^ - (filename)

B. Finding Cluster of Change Coupled (CC) classes:

CC classes are found by constructing a co-change matrix [13]. In this symmetric matrix, any cell, [A, B] and A≠B, represents of total changes of artifacts A or B, how many those changed together. Besides, in the cell [A, A] keeps track of how many times artifact A changed. Using appropriate support and confidence, the change coupled relation can be found among Java classes. Support represents how many times a class changed and confidence represents the likelihood, which means if there are 2 coupled artifacts, if one artifact changes, the probability that another artifact is going to change or not. In co-change matrix [A, A] represents support and confidence are represented by following Equation 1.

In Figure 2, there are 2 classes I and J. Among 5 changes of A and 3 changes of B, both of the co-changed 3 times. So the probability that if class A changes then class B will change or that confidence is obtained from equation 1 as $\frac{3}{5}$ or 60%. But if B changes then the confidence that A will change is $\frac{3}{3}$ or 100%.

CC classes were found by using different support and confidence in different works. Zimmermann et al used the support greater than 1 and confidence level 0.5 in their work [14]. However, Bavota et al considered elements that co-changing in at least 2% of the commits along with a confidence level of 0.8 [15]. Since, 0.8 confidence is high, in this paper 0.7 confidence level is used along with support 2 or more is considered. So in the considered 70% confidence and according to Figure 2, class J will have change coupled relation with I but not the opposite. This relationship is considered for classes found in FICs and the time period is taken from the 1st commit to the FIC. The source of finding this relationship, the co-change matrix is constructed by considering a n x n array, where n is the number of files or classes and co-changes of Java classes within the considered period is stored in that array. Then for each Java class, the change coupled relation with others are found based on considered support and confidence. Those other Java classes that have change coupled relation with a Java class is considered to form a cluster.

$$\begin{aligned} \text{Confidence}(A \rightarrow B) &= \frac{\text{support}(A \rightarrow B)}{\text{support}(A)} \\ &= \frac{\text{support}(A \cup B)}{\text{support}(A)} \end{aligned} \quad (1)$$

C. Commit History Analysis:

In this work, fixing changes are found by searching commits which later on leads to fix-inducing changes. Then using a commit window of 100 commits the entire commit history of source repository is traversed. However, the initial 20 commits were omitted for repository setup issues. After that commit window is used to traverse commits, i.e. 20-120, 120-220 and so on. For 100 commits in each window, FICs are analyzed for their contents. Figure 3 shows the entire process. After getting all FICs, these are sorted. Then traversing with commit

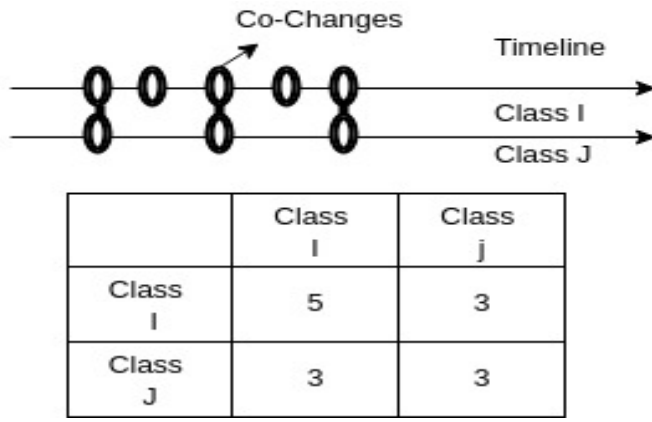


Figure 2: Example of Co-Change Matrix and Commit Timeline

frame of 100 commits, all FICs that falls within the frame can be found out.

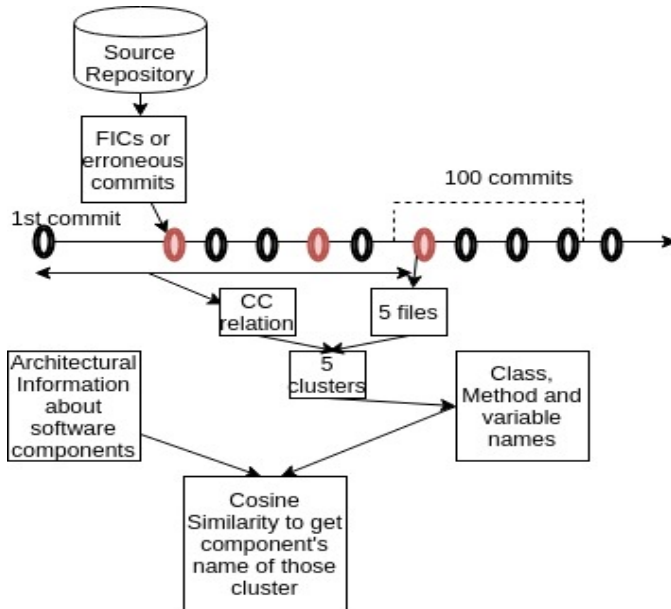


Figure 3: Commit History Analysis process to show the erroneous components.

In the contents of those FICs, firstly files with .java extension were searched. Then those files or classes were taken. For each of those class, cluster of classes found from the change coupled relation (with at least 2 support and 70% confidence) formed from 1st commit to that FIC are taken. Then that class in FIC along with its classes in change couple cluster are explored for their class, variable and method names. Then architectural information are taken from authenticated source and those information were first cleared of their stop words and then by applying the porter stemming algorithm their root words are taken. After that cosine similarity between those architectural information of different components and change coupled cluster information is taken to find out the most probable component for the cluster. After getting names for

each of these clusters, what percentage of total cluster belongs to which cluster can be easily known. Frequent appearance of the same component in commit history found by traversing with a commit window of 100 commits means that component is vulnerable to errors and responsible for costly corrective maintenance of the software system.

III. EXPERIMENTATION

This experiment is carried out in a virtual machine where operating system is Ubuntu 18.04 with 64 GB memory and 16 core CPU. For this experiment, among the popular Java repositories, 2 java repositories are selected for the study. These are as follows:

Repository Name	Source	Total Commits	Commits Analyzed	Number of Java classes	Lines of Code
Google Guava [16]	Github	4798	4020	3170	768858
jEdit [17]	Github	8000	8000	600	196194

TABLE I: General description of repositories.

Of the used 2 repositories, Google Guava is a source repository of library classes maintained by Google developers. This provides more functionalities than existing java collection framework and contains other features like hashing, graph, range etc. The commits of this repository started from June 2009 to the last commit updated in August 2018. In the case of jEdit, it is a text editor written in Java. In its source repository, commits started in 2001 and the last one is a patch commit in August 2019. By looking at commit history it can be said that the gradual development and maintenance is very slow in recent times.

Using the above mentioned repositories, the experiment is methodologically conducted. Firstly, Fixing changes were found by analyzing commit comments. After that, diffj [10] is used to find the lines that were modified or deleted from FCs parent commit. Since diffj ignores cosmetic and format changes, possibilities of finding false FICs are thus reduced. Then those identified lines are tracked by using Git Blame command to find the FIC commits where the last modifications are made. The total number of FCs and FICs that are found in both repositories are showed in Table II.

Repository Name	Fixing Commits	Fix-Inducing Commits
Google Guava	597	486
jEdit	2752	2270

TABLE II: Total Fix-Inducing Commits and Fixing Commits of each repository

The FICs are then sorted to find according to commit timeline. Then using a commit window of 100 commits, the entire history is traversed. For each FICs within the commit window, the .java files or classes are being collected. For each

class in FIC, a cluster of CC formed through the gradual development of the Change Couple relation throughout the lifespan of a project was analyzed. From the CC cluster of class in FIC and that class, information about classes, methods, and variables were collected. These are used to find cosine similarity with the description of components available from authenticated sources. Then of total clusters what percent belongs to a particular component for a particular 100 commit is found out.

Architectural information for google guava is collected from authenticated sources like in GitHub or their main website. In the case of jEdit, their main website is used to collect information about components. The obtained result mainly depends on this architectural information as cosine similarity is performed on this information.

IV. RESULT ANALYSIS

The results are obtained by conducting an experiment on the first 4000 commits of Google guava and 8000 commits of commits of jEdit. From the experiment, the name of components of google guava repository is obtained from [16] and that of jEdit is obtained from [17]. Table III and IV contains the name of the components considered.

Components	Description
Basic utilities	Deals with nulls, preconditions, common object methods, ordering, and throwables
Collections	It is an extension of JDK's collection system. It deals with immutable collections, new collection types, powerful collection utilities, and throwables.
Graphs	It mainly represents a graph, network and has structured data.
Caches	Local caching and support a wide variety of behaviors
Functional idioms	It is used to simplify the code greatly.
Concurrency	It contains powerful abstractions to write correct code.
Strings	It has useful string operations like joining, splitting and padding.
Primitives	It contains operations on primitives like int or char which is not provided by JDK
Ranges	It is an API to deal with both continuous and discrete ranges on comparable types.
I/O	It contains simplified I/O operations which specifically deals with I/O streams or files.
Hashing	It deals with hashing problems.
EventBus	It deals with publish-subscribe-style communication between components.
Math	It contains more optimized and tested math utilities, not provided by the JDK.
Reflection	It contains guava utilities for Java's reflective capabilities.

TABLE III: Information about components of Google Guava repository from github [16].

Components	Description
General	General features of the jEdit text editor providing writing and correcting facilities.
Source Code Editing	Dealing with source codes of different programming language.
Search and Replace	Different functionalities dealing with search.
File Management	It consists of everything related to files, like opening, editing, renaming and other such actions.
Customization	It consists of configurations to deal with users preference like customizable keyboard shortcuts etc.
Extensibility	Various plugins can extend the current abilities of jEdit to provide more functionalities.

TABLE IV: Information about components of jEdit repository from github [17].

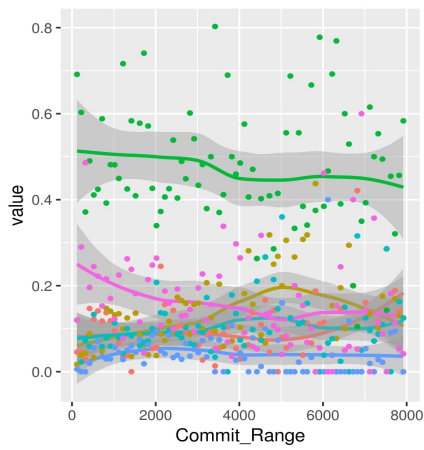
The name of the components are the features or modules found from the architectural information of corresponding sources. In Figure 4a the x-axis represents the commit number and y-axis represents the percentage of clusters obtained from FICs within the commit window of 100 commits. In Figure 4a, it is clearly visible that the file management part contributed a dominant portion to introduce errors into the software system. After it is found that file management, SourceEdit, and extensibility related classes are responsible for introducing errors into the system. jEdit being a text editor, surely works on file management will be more and it is expected to produce more errors. Next comes doing programming using jEdit, classes which manage it was producing more errors in the earlier phase of development which later on is replaced by extensibility. This might be because works on extensibility feature increased in later phase.

From Figure 4b, it is seen that seen that different components of the software system are affected at different times. However dominant top three are related to Math, Ranges, and Primitives. This may be associated with the nature of the project which is a library project. So mainly fixing occurs in classes when there are problems with continuous or discrete ranges, primitives with float or int values and calculations related to math.

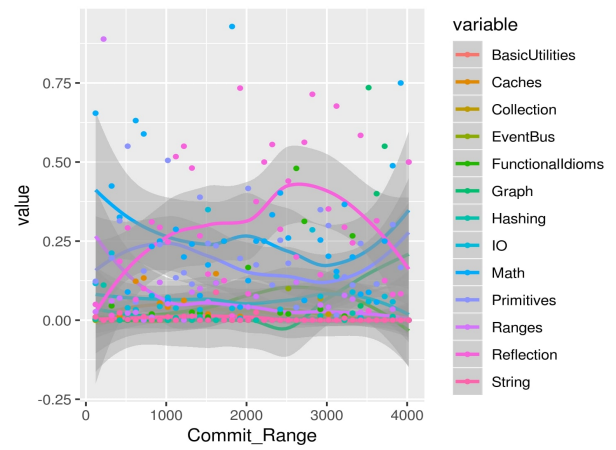
Through Figure 4a and 4b, any developer can understand classes of which components are producing more errors and thus can try to identify design flaws or code smells so that these can be addressed to improve the quality of software.

V. THREATS TO VALIDITY

The main factors for which construct validity might be threatened are described here. Firstly, only commit comments are searched to identify fixing changes without linking those to bug repositories. To create a link between fixing changes with bug report of the bug repository becomes a problem when the bug ids are not available. So for those cases linking with fixing changes might lead to unfair situation [18]. Besides, the main goal of this study is to find erroneous components. Secondly,



(a) Error History of components in jEdit Repository



(b) Error History of components in Google Guava Repository

Figure 4: Error History of components

there can be varying behavior in the contents of commits. Unrelated classes in bug fix can lead to wrong fix-inducing changes. But it is found that related works are committed together and 15% of all bug fixes to consist of multiple tangled changes [19]. Thirdly, all fixes may not be actually corrective maintenance [20]. However, using 10 random searches in FCs, only bug fixes are found. The main purpose of those FCs was corrective maintenance,

This work only considers java classes as Diffj, which is used for differencing modified and deleted source code lines between files of two commit version, can work only in java repositories. In the future extension, different types of project in different languages will be analyzed. Again, rather than error history, fixing history can also be obtained by considering the fixing changes. All of this information will be used to conduct further research in the fields of software architecture, decay and quality assurance.

VI. RELATED WORK

Evolution of software cannot be explained solely by structural dependency [21]. It is found that rather than structural dependency, change coupling plays a more effective role in fault proneness and are more relevant [22]. Historical information about CC classes can be used to predict further changes based on CC relation [23]. Similarly, based on CC relation, Zhou et al used Bayesian Network to predict changes [24]. Furthermore, Fluri et al used tree edit operations in AST to classify changes depending on how the change is made [25]. Rather than considering static measures, Arisholm et al proposed dynamic coupling measures by taking into account inheritance, polymorphism, and dynamic binding [26]. Moreover, whether or not frequent code changes represent code smell or design issues was investigated by Ratzinger et al and it is found that those changing software parts may be candidates for refactoring [27].

Frequent changes indicate unstable situations as changes do not satisfy the requirements and correctness expected from the

software system. Due to these frequent changes errors may be introduced in the system which is shown by D'Ambros et al [28] as change coupling measures have a strong relationship with software defects. These erroneous changes which introduced a bug or error in the system are called Fix-Inducing Change by Sliwerski et al [29]. Moreover, information of FICs and Fixing changes can be used for bug prediction [30] [31], localization [32] as well as to find out affected parts [33].

Very few works considered combining information from both change coupling and Bug/Errors. It is found that the change coupling relationship in recent commits is more correlated with recent FICs compared to commits from origin [34]. Furthermore, works of D'Ambros [7] [9] focused on analyzing software evolution and quality, and did not use the combined information to understand the error or maintenance history of software components.

VII. CONCLUSION

The main achievement of this work is to propose a methodology to analyze the erroneous components by using the change coupled relation at fix-inducing changes. Having knowledge of this information will help the software developers to find out which part of the system is continuously responsible for change and producing bugs. To do this, if separate information about architectural components is available then cosine similarity can be used. Otherwise, Latent Dirichlet Allocation can be used to find topics or probable components. However, in that case, manual labeling is required. From the obtained information, error-prone components can be easily identified. Then quality can be further enhanced by refactoring and re-engineering these error-prone components.

ACKNOWLEDGMENT

This research is supported by the fellowship from ICT Division, Ministry of Posts, Telecommunications and Information Technology, Bangladesh. No- 56.00.0000.028.33.002.19.3; Dated 09.01.2019. The virtual machine facilities used in this

research is provided by Bangladesh Research and Education Network (BdREN).

REFERENCES

- [1] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, 2005. [Online]. Available: <http://doi.acm.org/10.1145/1082983.1083147>
- [2] G. Antoniol, V. F. Rollo, and G. Venturi, "Detecting groups of co-changing files in CVS repositories," in *8th International Workshop on Principles of Software Evolution (IWPSSE 2005)*, 5-7 September 2005, Lisbon, Portugal, 2005, pp. 23–32. [Online]. Available: <https://doi.org/10.1109/IWPSSE.2005.11>
- [3] S. Levin and A. Yehudai, "Boosting automatic commit classification into maintenance activities by utilizing source code changes," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2017, Toronto, Canada, November 8, 2017*, 2017, pp. 97–106. [Online]. Available: <http://doi.acm.org/10.1145/3127005.3127016>
- [4] S. Kim, E. J. W. Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Trans. Software Eng.*, vol. 34, no. 2, pp. 181–196, 2008. [Online]. Available: <https://doi.org/10.1109/TSE.2007.70773>
- [5] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Software Eng.*, vol. 33, no. 1, pp. 2–13, 2007. [Online]. Available: <https://doi.org/10.1109/TSE.2007.256941>
- [6] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, 2014, pp. 172–181. [Online]. Available: <http://doi.acm.org/10.1145/2597073.2597075>
- [7] M. D'Ambros, M. Lanza, and M. Lungu, "Visualizing co-change information with the evolution radar," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 720–735, 2009.
- [8] A. E. Hassan and R. C. Holt, "Using development history sticky notes to understand software architecture," in *Proceedings. 12th IEEE International Workshop on Program Comprehension*, 2004. IEEE, 2004, pp. 183–192.
- [9] M. D'Ambros and M. Lanza, "Software bugs and evolution: A visual approach to uncover their relationship," in *Conference on Software Maintenance and Reengineering (CSMR'06)*. IEEE, 2006, pp. 10–pp.
- [10] Diffj. [Online]. Available: <https://github.com/jpace/diffj>
- [11] S. Kim, T. Zimmermann, K. Pan, and E. J. W. Jr., "Automatic identification of bug-introducing changes," in *21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006)*, 18-22 September 2006, Tokyo, Japan, 2006, pp. 81–90. [Online]. Available: <https://doi.org/10.1109/ASE.2006.23>
- [12] Z. Fabk, "Learn more about the history of a line with git blame," <https://zsoltfabok.com/blog/2012/02/git-blame-line-history/>, February 2012. [Online]. Available: <https://zsoltfabok.com/blog/2012/02/git-blame-line-history/>
- [13] T. Menzies, L. L. Minku, and F. Peters, "The art and science of analyzing software data: quantitative methods," in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2*, 2015, pp. 959–960. [Online]. Available: <https://doi.org/10.1109/ICSE.2015.306>
- [14] T. Zimmermann, S. Diehl, and A. Zeller, "How history justifies system architecture (or not)," in *Software Evolution, 2003. Proceedings. Sixth International Workshop on Principles of*. IEEE, 2003, pp. 73–83.
- [15] G. Bavota, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "An empirical study on the developers' perception of software coupling," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 692–701.
- [16] Guava architectural information. [Online]. Available: <https://github.com/google/guava/wiki>
- [17] jedit features. [Online]. Available: <http://www.jedit.org/index.php?page=features>
- [18] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: bias in bug-fix datasets," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2009, pp. 121–130.
- [19] K. Herzig and A. Zeller, "The impact of tangled code changes," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 121–130.
- [20] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*. ACM, 2008, p. 23.
- [21] M. M. Geipel and F. Schweitzer, "The link between dependency and cochange: Empirical evidence," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1432–1444, 2012.
- [22] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 864–878, 2009.
- [23] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," *IEEE Trans. Software Eng.*, vol. 31, no. 6, pp. 429–445, 2005. [Online]. Available: <https://doi.org/10.1109/TSE.2005.72>
- [24] Y. Zhou, M. Würsch, E. Giger, H. C. Gall, and J. Lu, "A bayesian network based approach for change coupling prediction," in *WCRE 2008, Proceedings of the 15th Working Conference on Reverse Engineering, Antwerp, Belgium, October 15-18, 2008*, 2008, pp. 27–36. [Online]. Available: <https://doi.org/10.1109/WCRE.2008.39>
- [25] B. Fluri and H. C. Gall, "Classifying change types for qualifying change couplings," in *14th International Conference on Program Comprehension (ICPC 2006)*, pages = 35–45, year = 2006, crossref = DBLP:conf/iwpc/2006, url = <https://doi.org/10.1109/ICPC.2006.16>, doi = 10.1109/ICPC.2006.16, timestamp = Mon, 22 May 2017 17:11:18 +0200, biburl = <https://dblp.org/rec/bib/conf/iwpc/FluriG06>, bibsource = dblp computer science bibliography, <https://dblp.org>.
- [26] E. Arisholm, L. C. Briand, and A. Foyen, "Dynamic coupling measurement for object-oriented software," *IEEE Transactions on software engineering*, vol. 30, no. 8, pp. 491–506, 2004.
- [27] J. Ratzinger, M. Fischer, and H. C. Gall, "Improving evolvability through refactoring," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, 2005. [Online]. Available: <http://doi.acm.org/10.1145/1082983.1083155>
- [28] M. D'Ambros, M. Lanza, and R. Robbes, "On the relationship between change coupling and software defects," in *16th Working Conference on Reverse Engineering, WCRE 2009, 13-16 October 2009, Lille, France*, 2009, pp. 135–144. [Online]. Available: <https://doi.org/10.1109/WCRE.2009.19>
- [29] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, 2005. [Online]. Available: <http://doi.acm.org/10.1145/1082983.1083147>
- [30] D. D. Nucci, F. Palomba, G. D. Rosa, G. Bavota, R. Oliveto, and A. D. Lucia, "A developer centered bug prediction model," *IEEE Trans. Software Eng.*, vol. 44, no. 1, pp. 5–24, 2018. [Online]. Available: <https://doi.org/10.1109/TSE.2017.2659747>
- [31] S. Shivaji, E. J. W. Jr., R. Akella, and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Trans. Software Eng.*, vol. 39, no. 4, pp. 552–569, 2013. [Online]. Available: <https://doi.org/10.1109/TSE.2012.43>
- [32] M. Wen, R. Wu, and S. Cheung, "Locus: locating bugs from software changes," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, 2016, pp. 262–273. [Online]. Available: <http://doi.acm.org/10.1145/2970276.2970359>
- [33] A. T. Misirli, E. Shihab, and Y. Kamei, "Studying high impact fix-inducing changes," *Empirical Software Engineering*, vol. 21, no. 2, pp. 605–641, 2016.
- [34] A. Z. Sadiq, M. J. I. Mostafa, and K. Sakib, "On the evolutionary relationship between change coupling and fix-inducing changes," in *15th International Conference on Evaluation of Novel Approaches to Software Engineering, ENSAE 2019*, in press.

Artifact Quality Assessment Plans Generation from Tailored Processes

Camila Hübner Brondani, Gelson Bertuol, Lisandra Manzoni Fontoura
Departamento de Computação Aplicada – DCOM
Universidade Federal de Santa Maria – UFSM
Santa Maria, Brazil
{chbrondani, gelson.bertuol, lisandramf}@gmail.com

Abstract— The production of quality software is a basic and essential Software Engineering goal. A software product quality assessment should be started at the early stages of a development process, to detect and correct problems before they propagate, making their correction more expensive. For that to be possible, it is necessary to assess the quality of each artifact generated during the development process, to allow the production of defect-free artifacts improving the final product quality. In this work, we propose an approach for the generation of quality plans during the tailoring of software process. When the user selects the quality practices to be used in a project, a set of activities satisfying those practices are inserted in the project's software process, along with their associated artifacts. Our goal is to define the quality assessment plans for these artifacts. The approach was validated through case studies of real projects in different companies, involving experts with large software development experience. The interviewees analyzed the approach and considered the proposal of this work as positive because it facilitates the definition of assessment plans, the plans are adequate to the selected criteria and that quality control during the process decreases rework.

Keywords— *Software Quality; Software Artifacts; Process Tailoring; Quality Practices.*

I. INTRODUCTION

It is consensual that software quality is highly influenced by the adoption of a software development process, this is mainly due to the management practices and the continuous pursuit for higher quality brought in by the processes, leading to software with fewer defects. The quality assurance activities aim to evaluate artifacts quality at each development stage, avoiding error propagation. Al-Kilidar et al. [1] state that, instead of trying to measure the quality of a software system as a whole, one should try to evaluate each intermediate software development product which, when combined, may provide for a broad idea about the whole system quality. However, even though quality is a recurrent issue in Software Engineering, most organization lack experts capable of defining their product's desired quality features. Furthermore, quality features definition and classification alone are not sufficient without a discussion about the means to reach those features and the relevant roles involved [2].

In this paper, it proposes a systematic approach for defining a quality assessment framework for artifacts generated or transformed by the activities that make up an tailored software process. The framework is composed by a metamodel, a knowledge base based upon the CMMI quality model [3], an assessment process and a supporting tool.

Briefly, the process engineer selects a set of quality practices that should be satisfied by the project. Practices are associated with activities that are recovered from the knowledge base and inserted in the project-specific process. For each activity, a set of artifacts is described, which are then evaluated regarding their quality elements using the ISO/IEC 9126 [4]. These, in turn, are described in a metamodel that links the assessment process to artifacts characteristics such as their purposes, their stakeholders, their corresponding methods and metrics. The final result is the definition of quality assessment plans for the artifacts described in the tailored software process.

These plans are based on the idea that a software product quality assessment may be started at the first stages of a software process [5]. Besides, they help the process engineers who need to customize the process for specific projects, allowing an organization to implement a set of quality practices from a stored knowledges base.

This paper is divided as follows: Section 2 presents the proposed framework, which includes: a) a metamodel, b) a knowledge base, c) an artifacts quality plan implementation process and d) a support tool. Section 3 shows the case studies. Finally, Section 4 presents our thoughts, conclusions and future work perspective.

II. THE DEFINITION OF AN ARTIFACT QUALITY ASSESSMENT PLAN

In this work, quality plans to assist evaluators, engineers and other stakeholders in the validation of software artifacts, are generated as a result of a process tailoring. For this, a set of tasks was proposed, as shown in Figure 1.

Initially, it is necessary to inform the characteristics of the process (task 1) and the process architecture that will be used in the adaptation (task 2). The next step is to select quality practices (task 3). Based on this information, a set of activities that meet the reported criteria will be retrieved from the knowledge base (task 4). The process engineer should select the activities that he wishes to include in the tailored process (task 5). When selecting the activities, the tailored process is created (task 6). In the next task, quality assessment plans for the artifacts are generated (task 7).

The tailoring is based on a process framework that integrates the necessary elements to instantiate tailored processes and to define quality plans. This framework is composed by a metamodel that presents the set of elements considered relevant for software artifacts quality assessment (described in section A), a knowledge base (described in section B), an assessment process (described in section C) and a supporting tool (described in section D).

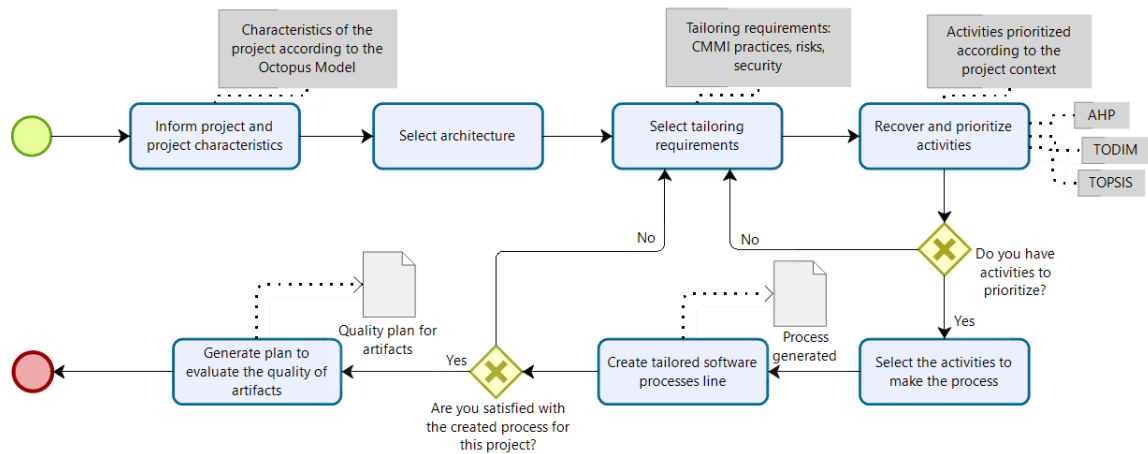


Fig. 1. Modeling Software Process Tailoring Approach to Quality Assessment of Artifacts

A. The Metamodel

The Quality Assessment of Artifacts in Process metamodel (QAPro-M) aims to provide and structure both the processes elements and the quality elements to be used in the assessments, helping the stakeholders to achieve a common vision of the quality requirements of a given project. It also allows for a structured decomposition of the elements, concepts and relationships necessary for this vision. The metamodel definition was based upon three basic requirements proposed by Trendowicz et al. [6]: flexibility, reusability and transparency.

The flexibility is associated to the software quality dependence on the context. The assessment model must be flexible enough to adapt to the different approaches resulting from each software project's organizational environment. For the present work, flexibility also refers to the differences among the very artifacts produced during a software development project phases. The documents, UML models, source code and other artifacts, each have their own particular characteristics. The assessment framework must then allow the evaluators to identify those characteristics and define which assessment proposals are best for each of them.

At the same time, reusability is associated to the preservation of knowledge from past experiences and its use on future projects, with a direct impact on development time and cost and, consequently, on the projects profitability. Naturally, reusability in an assessment project depends on projects similarity. Nevertheless, reuse may also refer to the measured data and to quality features and their relationships. Reuse also allows for model refinement, making it more precise and efficient.

Finally, an assessment model must provide a rational and transparent analysis about the relationships between quality features and sub-features, and about how they affect each other. For instance, the development team must be able to see how a class diagram modularization – that will later be used to define the database tables – allows for a better understanding of the software structure. However, it is a known fact that over-normalization may impair database performance, affecting the whole system. One solution to circumvent the transparency issues is to allow the interested parties themselves to define, consensually, the most relevant assessment metrics and methods, the ones that better represent each artifact, using the metamodel to reduce or eliminate any ambiguities and redundancies.

The QAPro-M, depicted in Figure 2, is composed by 23 classes. The metamodel central class is the **activity**. A set of activities from the same area is a **discipline**. Disciplines are distributed along interactive **phases** that sum up to form the development process **lifecycle**. An activity is composed by tasks, each **task** containing one or more roles. A **role** is a function or job carried out by a person in a project. The **project** class represents the projects defined for an **organization**. An organization may then have many projects, and each project may have many processes. This way, if a **process** does not fulfill the organization expectations, it may be evolved, the process new version being created based on the current one. When creating a new project or activity, it is possible to define a **situational context** for them. For project contextualization we used the Octopus Model that describes the following contextual factors: size, stable architecture, business model, team distribution, rate of change, system age, criticality and governance [7].

The process tailoring takes the activities and the tailoring criteria into account. Each activity has one or more **tailoring criteria**, whose function is to define which requirements may be satisfied by that activity instantiation, retrieving from the database the activities satisfying those criteria.

As a quality model, we chose to use CMMI, each organization can use the most appropriate model for their needs. Then, each **maturity level** is composed by a set of **process areas**. These, in turn, are composed by **specific goals**. Each specific goal is composed by a set of **specific practices**.

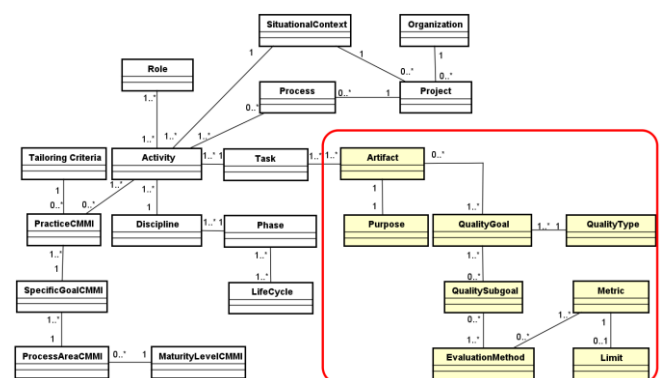


Fig. 2. Quality Assessment of Artifacts in Process - Metamodel (QAPro-M)

The **artifact** class represents the tasks outcomes. Artifacts are linked to a **purpose**, which identifies the artifact's intention and purpose inside the lifecycle, as well as the reasons why that artifact should be assessed.

The quality plan, on its turn, comprises the set of elements responsible for the artifact's assessment. The plans are related to the **quality goal** class. A quality goal represents an artifact's quality features or attributes of interest from a specific stakeholder. The metamodel allows for the quality goals to be identified by a **quality type**. ISO/IEC 9126 is an instance of a quality type, having 6 features, each feature subdivided into **quality subgoals**. The **evaluation method** identifies how a certain quality subgoal should be evaluated. These methods are generalized to allow for a specific methods and **metric**. Also, each metric defines a **limit** value as its acceptable value.

B. Knowledge Base

The first step towards defining the quality plan was to populate the knowledge base. Data was gathered from scientific literature, and included specific models and researches. The knowledge base is expandable, and may be grown based on the expert's experience from past projects.

In order to include quality-focused tailoring requirements, the CMMI quality model practices were incorporated to our approach. This way, the processes engineer may choose the desired organizational maturity level to be attained, the process area, the specific goal and, finally, the CMMI practices related to the goal to be reached by the process. Then the activities needed to satisfy the chosen practices are retrieved from the knowledge base in order to build the tailored process.

We started by analyzing the Requirements Management and Configuration Management process areas. We chose Requirements Management, since a project without well-defined and managed requirements has a far less probability of reaching its goals. Therefore, ensure the management of requirements is paramount to a project success. Likewise, Configuration Management is essential in order to produce quality software, since changes during development are inevitable. The practices described by CMMI were connected to activities and artifacts capable of satisfying each practice.

In order to organize the activities stored in the knowledge base in software processes, we used process architectures. Architectures are composed by the elements used to define a software process. They define the "skeleton" that a process must have, establishing the main elements and how they relate to each other [8]. In Figure 3, the architecture defined for the Requirements and Configuration and Change Management discipline can be viewed.

For each component, one or more activities are retrieved. The activities are prioritized using multi-criteria techniques such as AHP (Analytic Hierarchy Process), TODIM (an acronym in Portuguese for the Brazilian developed Iterative and Multi-criteria Decision Making Method) and TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution), taking into account the activity and the project's situational context.

Afterwards, a specific process for the project is composed by retrieving elements previously included in the

process architectures. So, using architecture allows for the retrieval of a set of activities prioritized accordingly to the project context and tailoring requirements. From the process architecture previously defined and the activities retrieved, prioritized and selected using the mathematical methods, it is possible to create the tailored process.

In previous work [9] we describe the process tailoring approach in detail. In this work, we extend this approach to incorporate process tailoring using quality criteria and to generate quality plans for assessing software artifacts.

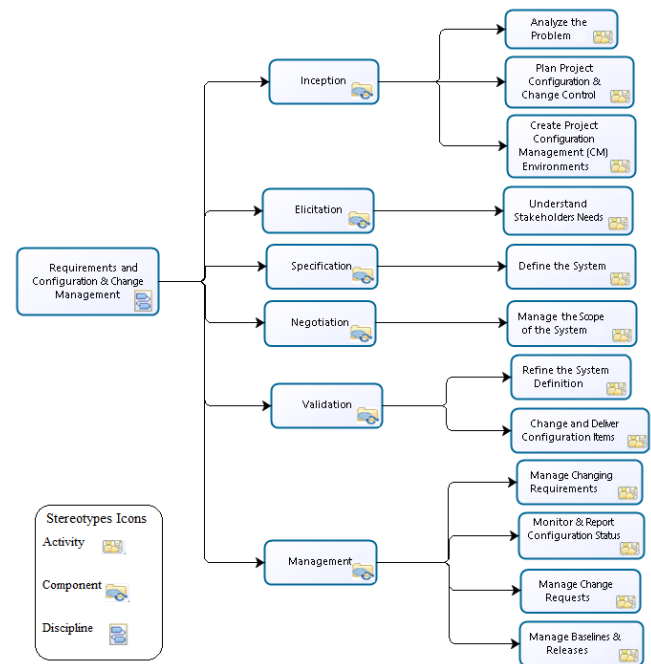


Fig. 3. Architecture for the Requirements and Configuration & Change Management discipline

C. Quality Plans Elaboration Process

The artifacts quality assessment process was build based on the ISO/IEC 14598 [10]. This standard was chosen both for its compatibility with the concepts laid out by the quality models and because it describes an assessment process for the quality features described by the ISO/IEC 9126 [4]. Some ISO/IEC 14598 sub-processes were adapted to reflect specific aspects of the software artifacts assessment.

The assessment process proceeds through the following steps: a) define the assessment requirements (defining what should be assessed); b) specify the assessment (selecting the goals or quality features related to the assessed object detailing assessment methods, metrics, limits and practices for each artifact); c) design the assessment (producing the assessment plan including the documentation of the procedures previously defined that will be used later to define the selected artifacts quality).

ISO/IEC 14598 last phase is the assessment execution, consisting of the products and its components inspection, measuring and testing, according to the assessment plan. As this is an execution-dependent task, not related to the process definition, it falls beyond the scope of this paper.

At the end of phase 3, the data describing the quality plans defined during the specific project assessment process

may be stored for use as a reference for future projects assessments.

As an example of the quality plan creation and the proposed approach workings, let us consider the user selected the CMMI practice “Obtain Commitment for the Requirements”. This practice belongs to the “Manage Requirements” Specific Goal, which in turn is part of the “Requirements Management” Process Area described in CMMI maturity level 2. The Typical Working Product for this practice is the “Documented commitments to requirements and requirements changes”.

Once this CMMI practice is selected, the knowledge base is searched for the activities that satisfy the practice. For instance, the “Manage Change Requests” and the “Manage Requirements Changes” activities from the RUP [11] disciplines “Configuration and Change Management” and “Requirements” will be retrieved and prioritized. After the activity’s prioritization, if this activity is selected to compose a tailored process, the goal would then be the generation of execution plans for the artifacts and process activities quality assessment.

Each activity has one or more input and output artifacts linked to a quality plan, describing how to assess them considering specially its features, objectives, stakeholders, assessment methods and metrics, in order to improve the final product’s quality. Figure 4 shows the metamodel instantiation for the “Change Request” artifact from the “Manage Change Requests” RUP activity assessment. This artifact’s objective is to document and follow the product’s change requests. The main quality goal is maintainability, based on the ISO/IEC 9126-proposed internal quality. From there the analyzability and changeability sub-goals may be explored. These goals and sub-goals may be assessed by specific methods and quantified by related metrics.

The quality plans main task is to organize the data structured by the metamodel, so the evaluators have a solid, clear and understandable reference when assessing the project’s artifacts, based on the quality goals they think are the most relevant for the final product.

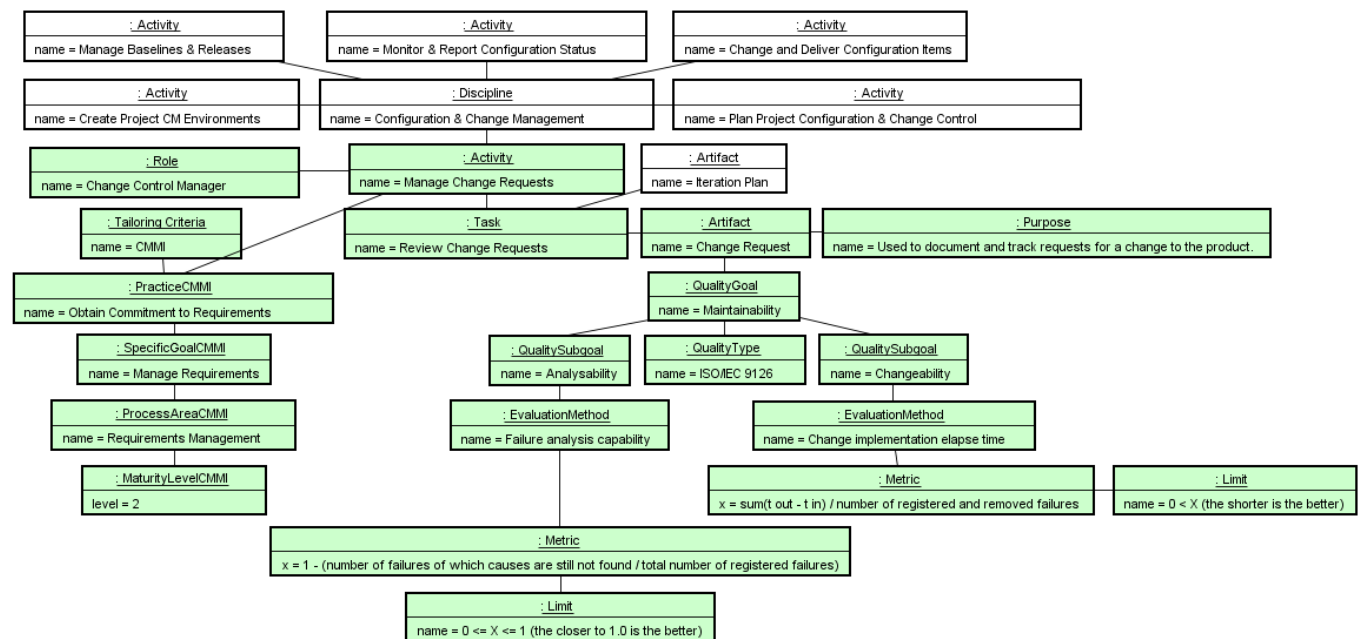


Fig. 4. Instance of the QAPro-M

D. QAPro System Support Tool

The Quality Assessment of Artifacts in Process (QAPro System) tool was developed from as extension of the MfTPt tool [9]. For the quality plans to be generated, instances of quality elements (described in QAPro-M) must have been defined and stored in the knowledge base.

The first activity of the systematic is the contextualization, for this we used the Octopus Model, allowing for the definition of values for each of its eight factors in the project (Figure 5-A). After that, it is necessary to define the desired architecture for the process tailoring.

The selection of requirements to be considered in the tailoring process are the Specific Practices of the Requirements Management e Configuration Management CMMI process area, shown in Figure 5-B.

Figure 5-C shows the architecture activities retrieval and prioritization according to the project context and the tailoring requirements, using the AHP, TODIM and TOPSIS methods. The process engineer selects the activities.

Finally, the last task consists in the generation of the plan to assess the created process artifacts. This step proceeds as follows: for each activity in the created process, a task is selected and then the artifact for which the quality assessment plan is to be obtained. The assessment specification for each selected artifact is then shown, with its objective, its quality goals and sub-goals, the quality type, the assessment method as well as the metric and the eventual limit value used for this artifact assessment (Figure 5-D).

This documentation will be used later by the quality analyst to define the selected artifacts quality. The idea behind the plan is to present quality elements in a clear and understandable way, to allow it to serve as a guide for evaluators during the process artifacts quality assessment.

However, the evaluators are free to modify any quality element according to the artifacts quality assessment needs. Thus, the flexibility and transparency requirements are kept during the assessment process.

Contextualize your project

Octopus Model

Size

☐ small
☒ medium
☐ large

Octopus Model

Stable Architecture

☐ Stable
☐ Changed
☒ New

Octopus Model

Business Model

☒ In house
☐ Commercial
☐ Large System Component

Octopus Model

Team Distribution

☒ Collocated
☐ Different teams
☐ Geographic

Octopus Model

Rate of Change

☐ More than 30
☐ From 10 to 30
☒ Less than 10

Octopus Model

Age of System

☒ New development
☐ Maintenance
☐ Legacy evolution

Octopus Model

Criticality

☒ Comfort loss
☐ Essential money loss
☐ Deaths

Octopus Model

Governance

☐ Dynamic/flexible
☒ Simple rules
☐ Mechanic/formal

CMMI - Requirements Management: Level 2

☒ SP 1.1 Understand Requirements
☐ SP 1.2 Obtain Commitment to Requirements

☒ SP 1.3 Manage Requirements Changes
☒ SP 1.4 Maintain Bidirectional Traceability of Requirements

☐ SP 1.5 Ensure Alignment Between Project Work and Requirements

CMMI - Configuration Management: Level 2

☐ SP 1.1 Identify Configuration Items
☐ SP 1.2 Establish a Configuration Management System

☒ SP 1.3 Create or Release Baselines
☒ SP 2.1 Track Change Requests

☐ SP 2.2 Control Configuration Items
☐ SP 3.1 Establish Configuration Management Records

☐ SP 3.2 Perform Configuration Audits

Prioritization

Activities	AHP	TODIM	TOPSIS
<input checked="" type="checkbox"/> Understand Stakeholder Needs	11,234 %	0,368	0,551
<input type="checkbox"/> Define the System	7,300 %	0,311	0,458
<input type="checkbox"/> Manage the Scope of the System	8,949 %	0,000	0,309
<input checked="" type="checkbox"/> Manage Changing Requirements	10,115 %	0,217	0,366
<input checked="" type="checkbox"/> Plan Project Configuration & Change Control	10,729 %	0,888	0,604
<input checked="" type="checkbox"/> Create Project Configuration Management (CM) Environments	10,115 %	0,217	0,366
<input type="checkbox"/> Manage Change Requests	9,105 %	0,278	0,436
<input checked="" type="checkbox"/> Monitor & Report Configuration Status	12,993 %	1,000	0,691
<input checked="" type="checkbox"/> Change and Deliver Configuration Items	10,955 %	0,802	0,604
<input checked="" type="checkbox"/> Manage Baselines & Releases	9,407 %	0,322	0,396

Quality Plans

Artifact	Description	Quality Type	Quality Goal	Quality SubGoal	Evaluation Method	Metric	Limit
Software Requirements Specification	This artifact captures the software requirements for the complete system, or a portion of that system.	External Quality - ISO/IEC 9126	Functionality	Functionality compliance	Functional compliance	X=1-(number of functionality compliance items specified that have not been implemented during testing / total number of functionality compliance items specified)	0<=X<=1 The closer to 1.0 is the better.
Software Requirements Specification	This artifact captures the software requirements for the complete system, or a portion of that system.	External Quality - ISO/IEC 9126	Usability	Understandability	Completeness of description	X=number of requirements described / total number of system requirements	0<=X<=1 The closer to 1.0 is the better.

Fig. 5. Quality Assessment of Artifacts in Process (QAPro System)

III. CASE STUDY

In order to validate this proposal, we carried out five case studies involving different real software projects from different companies. The case studies were carried out by three project managers, a compliance analyst and a systems analyst, so we could obtain a point of view from professionals in the field.

A. Case Study Configuration

The case studies were carried out in projects with different business domain. Each case study was divided in three phases:

Data collection: the participant (project member) filled a form identifying your profile, the project context and the tailoring requirements to be satisfied by the process.

Tailored process creation and quality plans generation: with the data informed and using the QAPro System support tool, the process was defined to meet the project's requirements and quality plans were generated for each artifact selected for the process. The results were sent to the participant for evaluation.

Interview and analysis: the participant experts were interviewed to evaluate the generated process and plans. The interview had 15 questions divided in 3 topics: software process, software quality and artifacts quality plans. Each interview lasted approximately 40 minutes, depending on the expert. After that, the results of the case studies were analyzed aiming to evaluate the applicability of the work.

B. Case Study – ASTROS Integrated Simulation System

To illustrate the validation process, this case study is described in detail. This project aims at the research and development of a virtual tactical simulator for military training. The interviewee was the project manager.

The project was characterized by the following attributes: a) Size (Medium); b) Team Distribution (Collocated); c) Criticality (Comfort Loss); d) Stable Architecture (New); e) Rate of Change (Less than 10); f) Governance (Simple rules); g) Business Model (In house) e h) Age of System (New development). The specific CMMI practices chosen were: Understand Requirements, Obtain Commitment to Requirements, Manage Requirements Changes, Identify Configuration Items, Establish a Configuration Management System, Create or Release Baselines, Establish Configuration Management Records, Perform Configuration Audits.

After the prioritization methods results analysis, the following activities were selected to compose the process: Plan project configuration & change control, Understand stakeholder needs, Define the system, Change and deliver configuration items, Monitor & report configuration status, Manage change requests, Manage baselines & releases.

Quality plans were suggested for the following artifacts: Configuration Management Plan, Stakeholder Request, Software Requirements Specification, Supplementary Specification, Workspace, Configuration Audit Findings, Change Request, Test Results, Test Log and Work Order.

These quality plans along with the tailored SPPrL were analyzed by the participant before the interview. The interviews results are discussed jointly in the next section.

C. Case Studies Discussion

The interview first part concerned the software development processes, centered around the following topics: presence of a well-defined process in the company, process tailoring use, agile or planned process use and software artifacts.

It was found that only one of the five projects has a development process with well-defined activities and artifacts, with a planning phase where the process compliance team analyses the project to verify its adherence

(or its lack of adherence) to agile practices. This is carried out through the use of checklist that evaluates, among other things, the project size, projected duration, team size, definitions and architecture patterns.

The goal of the interview second part, about software quality, was to discover if the organization uses any quality model, if a defective artifact can delay the project or raise its cost and if, by selecting CMMI practices as proposed here, there were improvements in the software process. The answers about quality models were unanimous, no project use them. All the participants stressed the difficult to adequate the model to the project's reality. As for defective artifacts, some participants mentioned cases from their own organizations showing that they may indeed delay and make projects more expensive.

As for the CMMI practices selection by the experts, all found it beneficial, as the foreknowledge of the practices to be followed allows for a better planning of the artifacts that would allow these practices to be attained. They found it a simple way to use the CMMI, with an intuitive approach that facilitates process creation. Also, the automated activities generation also saves effort and time, by replacing the need for a deeper model analysis.

The third topic goal was to understand the importance of the software artifacts quality assessment, which teams found the quality plans more efficient and if the artifact quality assessment plans format was satisfactory. This topic results analysis show that the proposed plans may be used by both large and small teams, with the adequate metrics varying according to the team and the project. One participant felt that the plans may be more important for large and distributed development teams, as the importance of documentation tends to grow, as well as the need for better artifacts.

All experts said that the proposed quality assessment plans allow for an artifact's assessment. They found the plans well-organized and easy to understand, and appreciated the plans clear separation of CMMI data, RUP data and artifact assessment.

IV. CONCLUSIONS AND FUTURE WORK

The benefits of developing quality software products are diverse, widespread and well known. However, the importance of evaluating the quality of these products goes beyond commercial or security issues, because it aims to provide qualitative and quantitative results on the quality of the software produced. In addition, provide the necessary feedback for the improvement of the software processes. But to be effective, these results must be understandable, trustworthy, and in keeping with the environment surrounding evaluations.

This work showed a framework for quality assessment of artifacts created or changed by activities forming a tailoring software process. For each process component, different activities may be selected, sharing a similar situational context and covering the desired quality practices to be used by the tailored process. These activities have artifacts, which in turn are associated to the quality plans defined here. The artifact assessment plan is elaborated taking into account the set of artifacts selected for the tailored process and reusing

instances of the quality metamodel. It should be noted that the generated quality plan serves as reference for the process engineers, who can, however, modify the plan if necessary. Besides that, the quality plans should evolve through time, along with the organization's maturity.

The approach validation was carried out using case studies. The participants agreed that the proposed approach to assess the software development generated artifacts is valid and relevant.

Future work includes the use of quality plans in real projects by monitoring quality throughout the project. To address one of the limitations observed, namely the use of activities found in planned processes, we also suggest the association of the quality practices to other groups of activities, such as the ones found in agile practices and methods. So it would be possible to create quality assessment plans with agile methods metrics, such as progress and productivity.

ACKNOWLEDGMENT

We thank the Fapergs (Fundação de Amparo à Pesquisa do Rio Grande do Sul) and the Brazilian Army for the financial support through the SIS-ASTROS Project (813782/2014), developed in the context of the PEE-ASTROS 2020.

REFERENCES

- [1] H. Al-Kildar, K. Cox, and B. Kitchenham, "The Use and Usefulness of the ISO/IEC 9126 Quality Standard," *International Symposium on Empirical Software Engineering*, pp. 126–132, 2005.
- [2] P. Mohagheghi, V. Dehlen, and T. Neple, "Towards a Tool-Supported Quality Model for Model-Driven Engineering," *Proceedings of the 3rd International Workshop on Quality in Modeling*, 2008.
- [3] SEI, "CMMI® for Development, Version 1.3," 2010.
- [4] ISO/IEC 9126, "Software Engineering - Product Quality," 2003.
- [5] T. L. Dubielewicz L, Hnatkowska B., Huzar Z., "Software Quality Metamodel for Requirement, Evaluation and Assessment," *ISIM'06 Conference*, pp. 115–122, 2006.
- [6] A. Trendowicz and T. Punter, "Quality Modeling for Software Product Lines," *7th Workshop on Quantitative Approach in Object-Oriented Software Engineering*, 2003.
- [7] P. Kruchten, "Contextualizing Agile Software Development," *Journal of Software: Evolution and Process*, vol. 25, no. 4, pp. 351–361, 2013.
- [8] A. S. Barreto, L. G. P. Murta, and A. R. C. da Rocha, "Software Process Definition: A Reuse-Based Approach," *Journal of Universal Computer Science*, vol. 17, no. 13, pp. 1765–1799, 2011.
- [9] W. G. Lorenz, M. B. Brasil, L. M. Fontoura, and G. V. Pereira, "Activity-based Software Process Lines Tailoring," *International Journal of Software Engineering and Knowledge Engineering*, vol. 24, no. 9, pp. 1357–1381, 2014.
- [10] ISO/IEC 14598, "Information Technology - Software Product Evaluation," 1999.
- [11] IBM Rational, "Rational Unified Process: Version 7.2," 2003.

Improve Language Modelling for Code Completion through Learning General Token Repetition of Source Code

Yixiao Yang
School of Software
Tsinghua University
Beijing, China
yangyixiaofirst@163.com

Xiang Chen
Beijing, China
kuailezhish@gmail.com

Jianguang Sun
School of Software
Tsinghua University
Beijing, China

Abstract—In last few years, to solve the problem of code completion, using a language model such as LSTM to learn code token sequences is the state-of-art method. However, tokens in source code are more repetitive than words in natural languages. For example, once a variable is declared in a program, it may be used many times. Other elements such as generic types in templates also occur repeatedly. It is important to capture token repetition of code. For example, if usage patterns of variables are not captured, there is little chance for a model trained on one project to predict the name of an unseen variable in another project correctly. Capturing token repetition of source code is challenging because not only the repeated token but also the place at where the repetition should happen must be both decided at the same time. Hence, we propose a novel deep neural model named *REP* to capture the general token repetition of source code. The repetitions of code tokens are modeled as edges connecting between repeated tokens on a graph. The *REP* model is essentially a deep neural graph generation model. The experiments indicate that the proposed model outperforms state-of-arts in code completion.

Index Terms—language model, code completion, code recommendation, code token repetition, deep neural graph generation

I. INTRODUCTION

In the past few years, language models have attracted the attentions of researchers and achieved a great progress in natural language processing tasks, such as machine translation [1] and text generation [2]. A statistical language model is a probability distribution over sequences of linguistic units such as characters or words. With the rapid growth of open-source code and the high-speed development of artificial intelligence, applying natural language processing techniques to source code has become a research direction. The problem of code completion is difficult and attracts a lot of attentions of researchers in the field of software engineering. Based on the already written code, the system of code completion can recommend the suitable code automatically for software developers to improve the efficiency of software development. Different from code synthesis based on natural languages [3], [4] which requires an explicit intention of code described

in natural languages, code completion is aimed at mining and learning the hidden intention of code to recommend suitable code automatically. Due to the predictability [5] of source code, taking source code as natural languages to build language models (n-gram, RNN or LSTM) to suggest code has made great progress in the field of code completion [6]–[11]. Among the previous works, deep learning models such as RNN or LSTM have been proved to be effective. In more subdivided fields of code completion, API usage pattern learning also attracts attention of a large amount of researchers [12], [13]. There are many differences between natural languages and source code. Source code is more regular than natural languages. There exist elements which may occur more than once in one fragment of source code, for example, variables or generic types in templates. Thus, taking source code as natural languages directly is not enough.

Based on the observation that elements of source code are highly repetitive, a new direction has been opened to capture the token repetition of source code to improve the performance of code completion. When predicting code, if an variable is unseen before, there is little chance to predict that variable correctly. If the usage pattern of an variable has been learned, the place at where the variable will be occurred next time can be decided. Then at next time, the name of that unseen variable can be predicted correctly by copying the name of that previously existed variable at the right place. Other elements such as templates also have the property of token repetition. For example, in Java languages, it is a common scenario to get the element from *ArrayList<T>* and the following code: *ArrayList<T> arr = ...initialization...; T t = arr.get(0);* is common in many programs. Note that T could be the name of any class. In a new project, if some strange class name such as *UnseenStrange* is in placed of T and *ArrayList<T>* becomes *ArrayList<UnseenStrange>*, through learning the repetitive patterns of source code, the right code: *UnseenStrange t = arr.get(0);* can be generated.

It is difficult to learn the token repetition of source code. There are two main challenges. The first challenge is that when predicting next token, we need to judge whether the next token

should be the repetition of some previously existed token. The second challenge is that if the next token is decided to be the repetition of some previously existed token, we need to decide which previously existed token should be repeated. If a program is huge and contains a large amount of tokens, it is hard to decide which token should be repeated. To address the two challenges, we propose a novel *REP* model to learn the token repetition of source code. The source code is parsed into a token sequence. This token sequence can be viewed as a linear graph as every token has an hidden incoming edge from its previous token. If two tokens in a token sequence are same, an edge is added between the repeated tokens. Then, a complex graph can be generated based on a token sequence. The extra added edge indicates the repetition of tokens. The task of *REP* model is to learn the edge connection on the generated graph. The experiments show that the proposed model outperforms state-of-art baselines. In summary, the contributions of this paper include:

- 1) A novel *REP* model is proposed to capture the general token repetition of source code;
- 2) Evaluations on four data sets (total 29.15MB) indicate that the proposed model outperforms the state-of-arts.

II. RELATED WORK

Models for code completion. The statistical language models have been widely used in capturing patterns of source code to solve the problem of code completion. In [5], source code was parsed into lexical tokens and the n-gram model was applied directly to suggest the next lexical token. In [14], a large scale experiments was conducted by using n-gram model and a visualization tool was provided to inspect the performance of the language model for the task of code completion. In SLAMC [6], based on basic n-gram model, associating code lexical tokens with roles, data types and topics was one way to improve the prediction accuracy. Cacheca [7] improved n-gram model by caching the recently encountered tokens in local files to improve the performance of basic n-gram model. Decision tree learning was applied to code suggestion, based on this, a decision tree model which integrates the basic n-gram [10] was proposed for source code. The work [15] abstracted source code into DSL and kept sampling and validating on that specially designed DSL until the good code suggestion was obtained. Deep learning techniques such as RNN, LSTM were applied to code generation model [8] [9] [11] to achieve a higher prediction accuracy. The work in [11] confirmed that LSTM significantly outperforms other models for doing token-level code suggestion. Given large amount of unstructured code, deep language models such as LSTM or its variants are the state-of-art solutions to the problem of code completion. All works described above are trying to solve the general code completion problem in which every token of code should be predicted and completed based on the context in a fixed or changeable length. There are also a lot of works paying attention to the API completion problem. Common sequences of API calls were captured with per-object n-grams in [12]. In [13], API usages was trained on graphs. Naive-Bayes was

integrated into n-gram model to suggest API patterns. The migrations of API are studied in [16]. The completion of API full qualified name is studied in [17].

Models for code synthesis. Another important research field to use language models is code synthesis. In recent years, translating text description into source code have achieved a great success. Seq2Seq [18], Seq2Tree [19] and Tree2Tree [20] models are proposed to handle the problem of code synthesis. The models for code synthesis are all based on the framework of neural machine translation. There are two modules named encoding module and decoding module in the framework of neural machine translation models. The encoding module encodes source sentences (trees) into fixed size vector. The decoding module decodes the fixed size vector generated by encoding module into sequences or trees. As discussed in [21], intuitively, Seq2Tree takes grammar of code into consideration but is still based on the framework of Seq2Seq. Although there are big differences between neural code translation and code completion, the aim of the decoding module in code synthesis is same as that of the language model. Both the aims are to generate the suitable code. So it is fair to compare our model with the decoding module in code synthesis model. Traditional decoding module of neural machine translation model uses standard LSTM directly, so there is no need to compare. But the recently proposed tree decoding module in Seq2Tree or Tree2Tree models deserves to be investigated for the code completion task. The decoding module in Tree2Tree model in most recent work [22] are extracted and compared with our *REP* model. There exists repetitiveness in the synthesized code. Our *REP* model could be taken as the decoding module for all code synthesis models directly. We will further investigate the performance of taking *REP* model as the decoding module in code synthesis tasks in the future work. On top of general code synthesis problems, API synthesis is also studied in [3], [4]. In the research fields of natural language processing, text summarization [23] is an important problem which is related to synthesis. One application of text summarization is to automatically generate the title of an article based on the content of that article. In source code processing, there is a similar problem: how to generate the name of a function according to the content of that function. This problem of function name generation has been addressed by extreme summarization of source code based on deep neural attention network in [24].

Models for code classification. For the problem of code classification or the problem of identifying code similarity, TreeNN [25], [26], TreeCNN [27], EQNET [28] or GNN (Graph Neural Network) [29] have been proposed. There exists huge differences between code classification problems and the code generation problems.

Models for robustness. To make the language model more robust, instead of improving the model structure directly, another research direction is to use different sampling schedules [30] or to generate adversarial examples [31] based on the training data to improve the robustness of the model.

Models for capturing token repetition of code. By com-

paring the related works mentioned all above, as far as we know, the *REP* model is the first deep neural model based on graph to capture the general token repetition of source code to help improve the solution to the problem of token-level code completion. The newly proposed model is also the first model to address problems of learning usage patterns of variables, templates and cloned code when doing code completion.

III. PROBLEM FORMULATION

Traditional language model processes tokens (words) one by one. The processing is based on a linear chain to handle each token in a token sequence. The data (token sequence) can be converted to the directed acyclic graph (DAG). Each node in the graph has only one incoming edge from its corresponding previous node. Such edge indicates the order of occurrence of tokens in a token sequence and we give that kind of edge a name *CommonEdge*. For example, if $token_n$ has one incoming edge from $token_{n-1}$, then $token_n$ should occur instantly after $token_{n-1}$. Then any token sequence could be represented by a simple graph. Follow this idea, to represent the repetition of tokens, we add an edge named *RepetitionEdge* between the repeated two tokens. For example, in a token sequence, if $token_m$ is same as $token_n$ where $m < n$, the special designed directed edge *RepetitionEdge* is added from $token_m$ to $token_n$. To see whether a token $token_n$ is repeated or not in a token sequence is just to see whether there is an edge linked to token $token_n$ from some previously existed token. Then, the problem of learning the general token repetition of source code reduces to the problem of learning the edge connection in a graph. The problem of code generation (code completion) reduces to the graph generation problem. In traditional language modelling, the edges between tokens do not need to be explicitly modeled. In our setting, the patterns of edge connections between tokens need to be explicitly modeled and learned. An example is shown in Figure 1. The

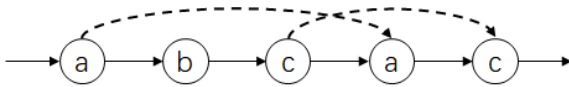


Fig. 1: Graph for Code and its Token Repetition

circle in Figure 1 represents the token. The edges marked as solid arrow (*CommonEdge*) show the order in which tokens are processed in a token sequence. The repeated tokens are connected by dashed arrow (*RepetitionEdge*). The *REP* model is still based on the framework of language model and can be taken as a complement to the language model. Tokens in a token sequence are predicted one by one. For each token, *REP* model additionally judge whether the currently predicted token should be the repetition of some previously existed token. As the language model may predict wrong content (wrong variable names or wrong templates), *REP* model could help correct the prediction through mixing the patterns of code repetitiveness together. In this setting, the *REP* model is designed to predict correctly the incoming *RepetitionEdge* of each token.

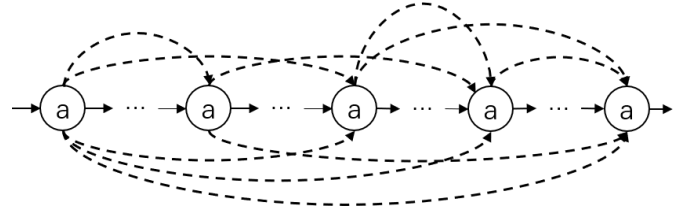


Fig. 2: Complex Graph for Code and its Token Repetition

Note that, one token may have more than one incoming edge if there exist two or more previous tokens with same content as current token in a token sequence. This complex situation is shown in Figure 2. Among all incoming edges (*RepetitionEdge*), if one edge could be predicted correctly, the prediction about the token repetition is right. So it is absolutely unnecessary to predict correctly all edges of kind *RepetitionEdge*. Learning to predict correctly one among all edges of kind *RepetitionEdge* is enough. To make the learning procedure easier, we could retain and learn only one incoming *RepetitionEdge*. In fact, considering only one of the all possible edges of *RepetitionEdge* not only meets the needs about deciding the token repetition but also makes the whole problem easier. Therefore, only the nearest two repeated tokens are connected by an edge (*RepetitionEdge*). Formally, for n th token $token_n$, only the k th token $token_k$ connects to $token_n$ where k is determined by

$$k = \max\{ i \mid token_i == token_n, i < n \} \quad (1)$$

The simplified graph corresponding to the graph in Figure 2 is shown in Figure 3. In Figure 3, only the edges connecting between the nearest two repeated tokens are retained. By removing unnecessary edges, now, every node has at most one incoming *RepetitionEdge* making the problem concise. When predicting next token, the whole problem of learning token repetition of source code is further divided into two sub-problems: 1. deciding whether there is a *RepetitionEdge* connected to next token; 2. if it is determined that there must be a *RepetitionEdge* connected to the next token, deciding which token is the source token of that *RepetitionEdge* (the next token should be same as the source token, in another word, the source token is the token to be repeated). In addition

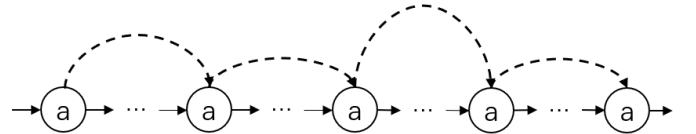


Fig. 3: Simplified Graph for Code and its Token Repetition

to the basic language model, learning the repetitiveness of source code (learning extra edge connections between tokens) can extract features of token repetition of source code to recommend code. The ability to learn token repetition can help us identify the usage patterns of variables, templates or cloned code. Details will be described in next section.

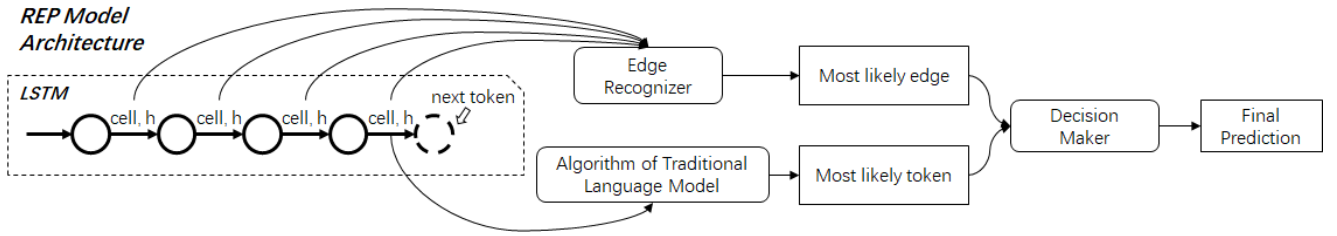


Fig. 4: Overall Architecture

IV. PROPOSED METHOD

Given the token sequences parsed from source code, our goal is to learn the general repetitiveness of tokens of source code to improve the performance of existing state-of-art language model. To accomplish this task, we design a novel *REP* model to learn the repetitiveness of source code tokens. Figure 4 demonstrates the overall architecture of our model. The basic part of *REP* model is the LSTM model. The LSTM model in *REP* generates the hidden feature vector (cell, h) for each token. When predicting the edge connection for $token_n$, the Edge Recognizer takes the hidden feature vector (cell, h) of $token_n$ and the hidden feature vector (cell, h) of each previously existed token to compute the probability for each possible incoming edge of $token_n$. The higher the probability, the more likely the edge should be added to the graph. For $token_n$, any edge of kind *RepetitionEdge* connecting from one of the previous tokens to $token_n$ is the candidate incoming *RepetitionEdge* of $token_n$. All possible candidate incoming edges (*RepetitionEdge*) of $token_n$ consist of edges of kind *RepetitionEdge* connecting from all the previous tokens to $token_n$. For example, for $token_3$, all candidate edges consists of two edges: 1. *RepetitionEdge* connecting from $token_1$ to $token_3$; 2. *RepetitionEdge* connecting from $token_2$ to $token_3$. The number of candidate edges (*RepetitionEdge*) is $n - 1$ for $token_n$. The Decision Maker is aimed at deciding whether the most likely edge predicted by Edge Recognizer should be really added or not. If the Decision Maker decides that there should be no edges (*RepetitionEdge*) connecting to the token being predicted, in this situation, the token predicted by the algorithm of the traditional language model will be taken as the final prediction result. If the Decision Maker decides that the next code token should be the one previously existed, in this situation, the source node (token) of the most likely edge will be thought to repeat at the position currently being predicted. So the token of the source of the recognized most likely edge will be taken as the final prediction result. To learn the general repetitiveness of source code tokens, when predicting a token, we need to take all previously existed tokens into consideration to decide whether or not one of the previously existed tokens should be repeated. The task becomes more and more challenging as the quantity of already predicted tokens becomes larger and larger. Traditional methods such as n-gram can only take limited quantity of tokens into consideration. There is little chance for those methods to discover the repetition of two tokens due to large

amount of other tokens between them. Deep learning methods scale well to high dimensional domains and have strong representational power. With the introducing of the deep neural networks, learning the general repetitiveness over a long token sequence becomes possible. In this subsection, we introduce our *REP* model which is based on the deep neural network.

A. LSTM in REP Model

The *REP* model is based on the LSTM model. LSTM model is applied to generate the state (cell and h) for every token in a sequence. Formally, the state generated for predicting $token_n$ by LSTM is referred to as $state_n$ ($cell_n$ and h_n). In rest of this section, the symbol $state_i$ ($cell_i$, h_i) refers to the state generated for predicting i th token $token_i$ by the LSTM. Traditional language model based on LSTM model uses h_n to compute the probability distribution of all candidate tokens to select the most likely token for $token_n$. The LSTM and its usage in traditional language model are omitted in this section. Please refer to [32] for the details of the algorithm of the traditional language model about how to use h_n which is generated by LSTM to compute the most likely token when predicting $token_n$. When predicting the edge connection for $token_n$, $h_n, h_{n-1} \dots h_1$ are used for computing the probability distribution of all candidate incoming edges of $token_n$. Details are shown in the following section.

B. Edge Recognizer in REP Model

Edge Recognizer is responsible for computing the probability distribution of all candidate incoming edges. Formal definition is as follows. The edge connecting from $token_m$ to $token_n$ where $m < n$ is referred to as $edge_{(m,n)}$. In rest of this paper, $edge_{(x,y)}$ refers to the *RepetitionEdge* connecting from x th token $token_x$ to y th token $token_y$. The probability of $edge_{(m,n)}$ is computed by:

$$P(edge_{(m,n)}) = \frac{h_m^T W h_n}{Z} \quad (2)$$

In Equation 2, Z is the normalization factor computed by:

$$Z = \sum_{m=1}^{n-1} h_m^T W h_n \quad (3)$$

The h_m is in the $state_m$ ($cell_m$, h_m) generated for m th token by LSTM. The h_n is in the $state_n$ ($cell_n$, h_n) generated for n th token by LSTM. In rest of this section, h_i is in the $state_i$ ($cell_i$, h_i) generated for i th token by LSTM. In Equation 2 and

3, W is the model parameter, h_m^T is the transposition of h_m . The training and predicting are all based on the probability of each candidate *RepetitionEdge* computed by Equation 2.

Training of Edge Recognizer. Remember that, in the section of Problem Formulation, each code will be converted into a token sequence, if $token_n$ is the repetition of some previously existed token, there must be an incoming edge of kind *RepetitionEdge* connecting to $token_n$. The source of that edge is the nearest previous token which is same as $token_n$. For $token_n$, we use the symbol: $near_n$ to refer to the position of that nearest token which is same as $token_n$. In another word, what we mean is that the nearest token (same as $token_n$) is the $near_n$ th token in the token sequence ($token_{near_n}$). According to the position of the source and the target, the incoming *RepetitionEdge* connecting from $near_n$ th token to n th token is referred to as $edge_{(near_n, n)}$. For $token_n$, we define a symbol $e_n \in \{0, 1\}$ which indicates whether there is an incoming *RepetitionEdge* connecting to $token_n$. If e_n is 1, this indicates that there is an incoming *RepetitionEdge* for $token_n$. If e_n is 0, this indicates that there is no incoming *RepetitionEdge* for $token_n$. If there is an incoming *RepetitionEdge* for $token_n$ in training examples, the source (token) of that incoming *RepetitionEdge* is referred to as $token_{near_n}$ which is the $near_n$ th token in the token sequence. The symbol $near_n$ refers to the position of the previously existed token which is nearest to $token_n$ and is same as $token_n$. To learn the edge connections in training examples, we follow the framework of Maximum Likelihood Estimation approaches: the probabilities of actually existed edges of kind *RepetitionEdge* in training examples should be as high as possible. Thus, the probability of the incoming *RepetitionEdge*: $edge_{(near_n, n)}$ (if there is one) for $token_n$ in training examples should be as high as possible. This is equivalent to minimize the following loss function. If there is no incoming *RepetitionEdge* for $token_n$, then $near_n$, $edge_{(near_n, n)}$ and $P(edge_{(near_n, n)})$ will be default meaningless values. This form avoids the using of condition judgment such as if-else to judge whether the incoming edge is existed or not and takes advantage of the high performance of GPU. With the probability of each edge, the loss function could be defined as (assuming that the quantity of code nodes in a data set is N):

$$\mathcal{L}_{ER} = \sum_{n=1}^N -\log(P(edge_{(near_n, n)})) * e_n \quad (4)$$

The symbol e_n indicates whether $token_n$ has an incoming *RepetitionEdge*. If there is no incoming *RepetitionEdge*, e_n is 0 and the final loss will exclude the loss for non-existent edges. The training objective of Edge Recognizer is to minimize the loss function 4.

Usage of Edge Recognizer in Prediction Phase. The edges with highest probabilities computed by Equation 2 are most likely to be added to the graph. In another word, the source of the edge with high probability has a high chance to be repeated. When predicting $token_n$, for the candidate incoming

RepetitionEdge with the k th largest probability, we use the symbol: src_k to refer to the position of the source of that k th most likely edge. In another word, the source of the k th most likely *RepetitionEdge* when predicting $token_n$ is the src_k th token ($token_{src_k}$). If $k = 1$, we can offer another formal definition of src_1 when predicting $token_n$:

$$src_1 = \arg \max_i P(edge_{(i, n)}) \quad (5)$$

When predicting $token_n$, if Decision Maker (described in the following subsection) decides that the $token_n$ should be the repetition of some previously existed token and top-k candidates are needed, $token_{src_1}$ (the source token of the edge with highest probability), $token_{src_2}$ (the source token of the edge with the second highest probability), ... and $token_{src_k}$ (the source token of the edge with k th highest probability) will be taken as the final recommendation. The top-k accuracy is computed by judging whether the desired $token_n$ exists in the candidates: $token_{src_1}$, $token_{src_2}$, ... and $token_{src_k}$.

C. Decision Maker in REP Model

The task of Decision Maker is to decide whether the edge with the highest probability computed by Edge Recognizer should be really added to the graph or not. When predicting $token_n$, Decision Maker is to decide whether there should be an edge of kind *RepetitionEdge* connecting to $token_n$. In the phase of predicting, the available information about *RepetitionEdge* is the probability of each candidate *RepetitionEdge* computed by Edge Recognizer for $token_n$. The probability that $token_n$ has an incoming *RepetitionEdge* ($token_n$ is the repetition of some previously existed token) is computed by:

$$P(token_n \text{ is repeated}) = \frac{h_{src_1}^T V_1 h_n}{h_{src_1}^T V_1 h_n + h_{src_1}^T V_2 h_n} \quad (6)$$

In the above equation, h_{src_1} is the h in state (cell, h) generated for the token at the position src_1 (src_1 th token) and $h_{src_1}^T$ is the transposition of h_{src_1} . The src_1 is defined in Equation 5. V_1 and V_2 are model parameters.

Training of Decision Maker. If there is an incoming edge in training example for $token_n$, $P(token_n \text{ is repeated})$ defined in Equation 6 should be maximized. Otherwise, the value: $1 - P(token_n \text{ is repeated})$ should be maximized. This corresponds to minimize the following loss. The loss function of Decision Maker is defined as:

$$\mathcal{L}_{DM} = \sum_{n=1}^N (-\log(P(token_n \text{ is repeated}) * e_n - \log(1 - P(token_n \text{ is repeated})) * (1 - e_n))) \quad (7)$$

The training objective of Decision Maker is to minimize the loss function 7. The final loss $\mathcal{L} = \mathcal{L}_{ER} + \mathcal{L}_{DM}$. To minimize the final loss \mathcal{L} is equivalent to minimize \mathcal{L}_{ER} and \mathcal{L}_{DM} separately.

Usage of Decision Maker in Prediction Phase. When predicting $token_n$, if $P(token_n \text{ is repeated})$ is greater or equal to 0.5, the recommendation result generated by Edge

Recognizer (described in the previous subsection) should be taken as the final result. Otherwise, if $P(token_n \text{ is repeated})$ is less than 0.5, the prediction result generated by the algorithm [32] of the traditional language model will be taken as the final prediction result.

D. Advanced REP Model

In previous subsections, there is only one LSTM model described in *REP*. To improve the expressiveness of the *REP* model, we could use two LSTM models, one for computing probabilities of edges in *REP*, one for computing the prediction result of the standard language model. In another word, the LSTM used for computing the prediction result of standard language model is the LSTM isolated from the LSTM model described in *REP*. Also, the embeddings of tokens involved in those two LSTM models could be isolated. In conclusion, we use one LSTM model as the standard language model and use another LSTM model to compute the probabilities of edges in *REP*. By doing so, the parameters have doubled in size and the performance of the *REP* model could be further improved. Actually, the *REP* model used in experiments is the advanced version just described in this subsection.

V. IMPLEMENTATION

Source code [33] which contains all data sets and all implementations of all models mentioned in experiments has been public. The implementation of the model is based on the deep learning platform TensorFlow. Apart from the parameters of the LSTM used in *REP* model, the other parameters of *REP* model are W in equation 2 and V_1, V_2 in equation 6. Adam optimizer in TensorFlow is used to automatically decide learning rate and momentum in training phase. The gradient is clipped by global norm. Parameters about clipping norm are set to default values offered by TensorFlow. The representation size (alias as embedding size or feature size) for one token is 128. Once the embedding size for one token is decided, sizes of all other parameters which participate in the calculation with the embedding of tokens can be decided successively. All models keep training until the accuracy on validation set does not exceed the optimal value for 50 epochs. All the training examples are trained one by one. The models are running on the computer with the setting: Windows 10 64 bit OS, Intel i7-6850k CPU, 32G memory and one Geforce GTX 1080 Ti GPU. Every function in source code will be extracted and tested. The code completion system will start at the beginning of a function to try to complete each token of the function one by one. The accuracy is the average of the prediction accuracy of each token of each function in test set. To generate tokens for the source code, some works [5] parse the code into lexical units through splitting the code by white space or other predefined separators such as $+$, $($, $:$ or $;$. The implementation of such parser may vary greatly. Different separators in use lead to different token sequence of source code. Thus, to make the token generation for code unified, the token sequence is generated in the following steps. The code is parsed into the abstract syntax tree (AST) through Eclipse JDT

at first. Then, the AST is traversed in pre-order. The content of each encountered node is pushed back onto a sequence. Now, the token sequence has been generated. Token repetition learning is also based on the token sequence generated in this way. Generating token sequence based on AST could also make it fair to compare the sequential model such as LSTM with tree models such as Tree2Tree model as the contents to be predicted are same in this setting.

VI. EXPERIMENT

Without loss of generality, the most widely used programming language: Java is chosen to conduct experiments. Four data sets are provided for evaluations. For each data set, the source code in that data set is divided into training set, validation set and test set in the proportions 60%, 15%, 25%. Follow the one billion word benchmark [34], similarly, the 0.15% least frequently occurred code tokens in our training set, all unseen tokens in validation set and test set are marked as *UNK*.

Data Sets. Famous open-source projects are used in experiments. As there are small functions which contain only one or two statements in those projects, we do some filtering to the code. The filtering steps are as follows: 1. Give each Java file a score (to get the score, divide the number of all tokens in all functions in the java file by the number of all functions in that java file); 2. Sort Java files according to the score from large to small and extract the first 85% Java files to blend into a data set. Details are shown in Table I. The fourth column headed by *Vocabulary* in Table I means the total number of unique tokens on the data set. The symbol DS refers to data set. For example, the DS1 refers to data set 1. The original size of Apache Lucene is 98.8MB and is too huge, so only the core module and analysis-common module are extracted into the data set.

TABLE I: Data Sets

	From Project	Size	Vocabulary
DS1	Log4J	1.76MB	6455
DS2	Maven	3.25MB	10516
DS3	FindBugs (GitHub version)	8.54MB	27573
DS4	Lucene (core & analysis-common)	15.6MB	55244

Research Questions. To demonstrate the ability of the proposed model, one research question is answered:

RQ1: Could *REP* model achieves better prediction accuracy than other state-of-art methods under all the data sets?

To evaluate the performance of our model, the state-of-art baselines: LSTM and tree decoding module in Tree2Tree [22] are compared with the *REP* model. The tree decoding module in other Tree2Tree models [35] is just LSTM model with slightly changes (one LSTM for decoding content of node, one LSTM for decoding the tree structure of node) and is irrelevant to this research because we only care about predicting the content of node in this research. Table II shows the prediction accuracy of different models evaluated on the test set of each data set. The top-k accuracy (value is in percentage, % is omitted to save the space) is computed for evaluating

TABLE II: Evaluation Result

	LSTM						Tree2Tree						REP					
	top1	top3	top6	top10	mrr	enpy	top1	top3	top6	top10	mrr	enpy	top1	top3	top6	top10	mrr	enpy
DS1	45.4	59.3	63.7	66.3	0.53	6.7	34.7	54.3	61.5	64.7	0.45	5.9	48.6	63.0	68.1	70.9	0.57	2.8
DS2	48.0	60.7	65.2	67.7	0.55	6.8	36.7	54.4	59.8	63.0	0.46	5.8	52.6	66.3	71.2	73.8	0.60	3.3
DS3	34.5	49.2	55.2	58.6	0.43	7.7	31.8	50.0	56.4	59.2	0.39	6.9	44.7	59.9	66.3	69.9	0.53	5.8
DS4	48.9	63.7	69.8	73.3	0.57	3.6	40.0	58.0	65.1	69.3	0.50	4.0	53.9	69.1	75.3	78.6	0.62	1.7

the model performance. When predicting next node, we rank all candidate tokens according to probabilities (computed by model) from large to small. The token with higher probability has smaller rank. The token with the highest probability has rank 1. The value in column headed with *mrr* is the average of the reciprocal of rank of the token. This metric indicates the overall prediction performance of the model. The larger the *mrr*, the better the performance of the model. The *enpy* headed column shows the entropy (*log* value of the perplexity) of the model. The smaller the entropy, the better is the model.

As can be seen from the data, the *REP* model outperforms all other state-of-art models in all 4 data sets. Especially for the top-1 accuracy, *REP* model achieves averagely 17.2% improvement compared to LSTM, achieves averagely 39.7% improvement compared to Tree2Tree. From the result, we can conclude that capturing the token repetition of source code can improve existing LSTM model which has solved the problem of gradient vanishing and gradient exploding to capture the long term memory. Source code is diverse. After randomly checking files in test set and training set, we have discovered that there is little chance for the exact same code appears in both the training set and test set. As argued in [30], if some sub-sequences or patterns in the context are unseen in training data, the discrepancy between training and inference could cause the system fail to predict the right result. This problem is named as *exponential bias* and *REP* model could solve that problem to some extent.

The tree decoding module in Tree2Tree [22] performs worse than LSTM. From this experiment, tree decoding module in Tree2Tree is strongly dependent on the attention mechanism used in its encoding module. Without the cooperation from the encoding module, the performance of a single decoding module is worse than standard LSTM. The reason has been carefully analyzed. In Tree2Tree model [22], trees need to be converted into binary trees. Converting a general tree to a binary tree needs to add more nodes makes the prediction task more difficult. The decoding module decodes the binary tree in the way that the left child node and right child node of one node will be predicted at the same time. This decoding procedure indicates that the right child is predicted without the information of the left child and vice versa. In the meanwhile, LSTM model predicts one by one meaning that the right child is predicted with the information of the left child (according to pre-order traversal of tree). The less use of the context information causes the lower prediction accuracy of tree decoding module in Tree2Tree [22]. On the other hand, the less use of the context could reduce the impact of the unseen data in context. When encountering a large amount of unseen

data, Tree2Tree could gain good generalization ability. That is why Tree2Tree performs better than LSTM in entropy on the first three data sets. Even so, Tree2Tree still performs worse than *REP* in entropy. As can be seen from the entropy (*log* value of the perplexity), *REP* achieves nearly half the entropy compared to that of other models. This experimental result is dramatically encouraging. Thus, by taking all conditions into account, *REP* outperforms other models.

RQ2: In which scenarios can *REP* model achieve better results than other models and what features of scenarios bring *REP* model to the better performance?

In the investigation of the prediction result of the experiment, we discover that *REP* model is good at predicting unseen data especially for unseen variable names or unseen type names. In our setting, unseen token in validation set or test set will be replaced with UNK. Both LSTM model and Tree2Tree model perform badly in distinguishing between UNK and other tokens when the number of tokens is huge. In *REP* model, by recognizing the hidden relationships of token repetition, another point of view could be offered to us to decide which is the most likely token for the next. If the model can confirm that the next token should be the repetition of some previously existed token, we could take the previously existed token as the final prediction result no matter the previously existed token is UNK or other tokens. Note that, for a token sequence of which the length is often less than 1000, the number of the previously existed tokens is at most 1000. In this setting, the *REP* model only needs to distinguish between at most 1000 tokens to decide the token repetition. In the meanwhile, for large projects, the number of total tokens is often greater than 6000, which means that the standard LSTM-based language model needs to distinguish between at least 6000 tokens to decide which is the most likely token for the next. Obviously, the task of recognizing token repetition is much easier than the task of the standard LSTM-based language model. That is the one factor why *REP* model could perform better than standard LSTM-based language model.

Limitation and Future Work. In experiments, only four Java projects are used, more projects could make the results of experiments more solid. In all projects of different sizes, the proposed model has achieved better results than all other models, which can prove the effectiveness of the model to a certain extent. The proposed method is not limited to the Java language and can be extended to other languages such as Python and C++. The extra work needed to do is to design Python parser or C++ parser to parse the code into the corresponding token sequence. The proposed model could be applied to the generated token sequence. In the future,

the performance of the proposed model could be further investigated on different languages such as Python or C++. If the code corpus is large, there would be a lot of tokens. The existence of a large number of tokens can cause trouble to apply this technique to industrial scenarios. The techniques which are designed to minimize the number of tokens could be applied to further reduce the total tokens to improve the model performance.

VII. CONCLUSION

In this paper, a novel *REP* model is proposed to capture the general token repetition of source code to improve the prediction accuracy of standard language model. The experimental results on huge data sets confirm that capturing the general token repetition of source code by *REP* model successfully improves current methods and makes a step forward for the problem of code completion.

REFERENCES

- [1] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *Computer Science*, 2014.
- [2] D. Pawade, A. Sakthapara, M. Jain, N. Jain, and K. Gada, "Story scrambler-automatic text generation using word level rnn-lstm," *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 10, no. 6, pp. 44–53, 2018.
- [3] T. Nguyen, P. C. Rigby, A. T. Nguyen, M. Karanfil, and T. N. Nguyen, "T2api: synthesizing api code usage templates from english texts with statistical translation," in *ACM Sigsoft International Symposium on Foundations of Software Engineering*, 2016, pp. 1013–1017.
- [4] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API learning," in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, 2016, pp. 631–642. [Online]. Available: <https://doi.org/10.1145/2950290.2950334>
- [5] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. T. Devanbu, "On the naturalness of software," in *ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, 2012, pp. 837–847. [Online]. Available: <http://dx.doi.org/10.1109/ICSE.2012.6227135>
- [6] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "A statistical semantic language model for source code," in *ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, 2013, pp. 532–542. [Online]. Available: <http://doi.acm.org/10.1145/2491411.2491458>
- [7] Z. Tu, Z. Su, and P. Devanbu, "On the localness of software," in *The ACM Sigsoft International Symposium*, 2014, pp. 269–280.
- [8] M. White, C. Vendome, M. Linares-Vasquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *Ieee/acm Working Conference on Mining Software Repositories*, 2015, pp. 334–345.
- [9] H. K. Dam, T. Tran, and T. T. M. Pham, "A deep language model for software code," in *FSE 2016: Proceedings of the Foundations Software Engineering International Symposium*. [The Conference], 2016, pp. 1–4.
- [10] V. Raychev, P. Bielik, and M. T. Vechev, "Probabilistic model for code with decision trees," in *OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, October 30 - November 4, 2016*, 2016, pp. 731–747. [Online]. Available: <http://doi.acm.org/10.1145/2983990.2984041>
- [11] V. J. Hellendoorn and P. Devanbu, "Are deep neural networks the best choice for modeling source code?" in *Joint Meeting on Foundations of Software Engineering*, 2017, pp. 763–773.
- [12] V. Raychev, M. T. Vechev, and E. Yahav, "Code completion with statistical language models," in *PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, 2014, p. 44. [Online]. Available: <http://doi.acm.org/10.1145/2594291.2594321>
- [13] A. T. Nguyen and T. N. Nguyen, "Graph-based statistical language model for code," in *ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, 2015, pp. 858–868. [Online]. Available: <http://dx.doi.org/10.1109/ICSE.2015.336>
- [14] M. Allamanis and C. A. Sutton, "Mining source code repositories at massive scale using language modeling," in *MSR '13, San Francisco, CA, USA, May 18-19, 2013*, 2013, pp. 207–216. [Online]. Available: <http://dx.doi.org/10.1109/MSR.2013.6624029>
- [15] V. Raychev, P. Bielik, M. T. Vechev, and A. Krause, "Learning programs from noisy data," in *POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, 2016, pp. 761–774. [Online]. Available: <http://doi.acm.org/10.1145/2837614.2837671>
- [16] T. D. Nguyen, A. T. Nguyen, H. D. Phan, and T. N. Nguyen, "Exploring api embedding for api usages and applications," in *IEEE/ACM International Conference on Software Engineering*, 2017.
- [17] H. Phan, H. Nguyen, N. Tran, L. Truong, A. Nguyen, and T. Nguyen, "Statistical learning of api fully qualified names in code snippets of online forums," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 632–642.
- [18] S. Iyer, I. Konstantas, A. Cheung, and L. Zettlemoyer, "Mapping language to code in programmatic context," *arXiv preprint arXiv:1808.09588*, 2018.
- [19] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," *arXiv preprint arXiv:1704.01696*, 2017.
- [20] M. Drissi, O. Watkins, A. Khant, V. Ojha, P. Sandoval, R. Segev, E. Weiner, and R. Keller, "Program language translation using a grammar-driven tree-to-tree model," *arXiv preprint arXiv:1807.01784*, 2018.
- [21] J. Sedoc, D. Foster, and L. Ungar, "Neural tree transducers for tree to tree learning," 2018.
- [22] X. Chen, C. Liu, and D. Song, "Tree-to-tree neural networks for program translation," *arXiv preprint arXiv:1802.03691*, 2018.
- [23] J.-g. Yao, X. Wan, and J. Xiao, "Recent advances in document summarization," *Knowledge and Information Systems*, vol. 53, no. 2, pp. 297–336, 2017.
- [24] M. Allamanis, H. Peng, and C. Sutton, "A convolutional attention network for extreme summarization of source code," in *International Conference on Machine Learning*, 2016, pp. 2091–2100.
- [25] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng, "Semantic compositionality through recursive matrix-vector spaces," in *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*. Association for Computational Linguistics, 2012, pp. 1201–1211.
- [26] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.
- [27] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 1287–1293.
- [28] M. Allamanis, P. Chanthirasegaran, P. Kohli, and C. Sutton, "Learning continuous semantic representations of symbolic expressions," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 80–88.
- [29] X. Xu, L. Chang, F. Qian, H. Yin, S. Le, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," 2017.
- [30] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," 2015.
- [31] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *AAAI*, 2017, pp. 2852–2858.
- [32] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.
- [33] "The source code of models in the experiments and all data sets," <https://www.dropbox.com/s/28p8j44zdc78ob4/REP.zip>.
- [34] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One billion word benchmark for measuring progress in statistical language modeling," *arXiv preprint arXiv:1312.3005*, 2013.
- [35] D. Alvarez-Melis and T. S. Jaakkola, "Tree-structured decoding with doubly-recurrent neural networks," 2016.

Improve Language Modelling for Code Completion by Tree Language Model with Tree Encoding of Context

Yixiao Yang
School of Software
Tsinghua University
Beijing, China
yangyixiaofirst@163.com

Xiang Chen
Beijing, China
kuailezhish@gmail.com

Jianguang Sun
School of Software
Tsinghua University
Beijing, China

Abstract—In last few years, using a language model such as LSTM to train code token sequences is the state-of-art to get a code generation model. However, source code can be viewed not only as a token sequence but also as a syntax tree. Treating all source code tokens equally will lose valuable structural information. Recently, in code synthesis tasks, tree models such as Seq2Tree and Tree2Tree have been proposed to generate code and those models perform better than LSTM-based seq2seq methods. In those models, encoding model encodes user-provided information such as the description of the code, and decoding model decodes code based on the encoding results of user-provided information. When applying decoding model to decode code, current models pay little attention to the context of the already decoded code. According to experiments, using tree models to encode the already decoded code and predicting next code based on tree representations of the already decoded code can improve the decoding performance. Thus, in this paper, we propose a novel tree language model (TLM) which predicts code based on a novel tree encoding of the already decoded code (context). The experiments indicate that the proposed method outperforms state-of-arts in code completion.

Index Terms—code completion, tree language model, tree encoding of context

I. INTRODUCTION

Taking highly repetitive and predictable [1] source code as natural languages to build language models (n-gram, n-gram with Bayes or decision tree, RNN, LSTM) to suggest code fragments has made great progress [2]–[9]. However, sequential models does not explicitly model the tree structure. In the meanwhile, tree classification models such as TreeNN [10] and TBCNN [11] have more powerful ability in capturing the characteristics of trees. The works in [10] have confirmed that tree models perform better than sequential models in classification tasks. On the other hand, it is hard to adapt the code classification models as the code generation models. This paper addresses this challenging problem and demonstrates the special property of the proposed model.

In code synthesis models, encoding module is used for encoding the user-provided information, decoding module is used for decoding the desired code from the encoding of

the provided information. In decoding module, we creatively use encoding model to encode the already decoded code to improve the performance of the decoding model. A novel tree decoding model (tree language model) which consists of Encoding model and Decoding model is proposed. In the rest of this paper, the term encoding model refers to the model which is used for encoding the already decoded code (context) in decoding procedure, not the one used for encoding the provided information in code synthesis tasks. Encoding model is responsible for generating the representation for a tree or a sub-tree. Decoding model is designed to traverse the syntax tree to accumulate encoding of encountered sub-trees to predict the next code. We propose a novel encoding model based on two-dimensional LSTM to generate better representations for trees. The experiments indicate that the proposed model outperforms state-of-arts. In summary, the contributions of this paper include: 1) A framework is proposed to predict next code based on the accumulated representations of sub-trees in an AST. 2) A new encoding model based on two-dimensional LSTM is designed to generate representations for trees.

Related Work: The statistical n-gram language model has been widely used in capturing patterns of source code. The n-gram model was applied to lexical code tokens in [1]. In SLAMC [2], they improved n-gram by associating code tokens with topics. In cacheca [4], they improved n-gram with caching recently appeared tokens in local files to improve prediction performance. In [12], source code was abstracted into DSL and had been sampled and validated until the good code suggestion was obtained. Deep learning techniques such as RNN, LSTM were applied to code generation model [6] [8] [9] to achieve a higher prediction accuracy. Decision tree was applied to code suggestion relying on a hybrid language model presented by [7]. In [5], code was trained on graphs. Naive-Bayes was integrated into n-gram model to suggest API usages. Seq2Seq [13], Seq2Tree [14] models were proposed to translate text description into the abstract syntax tree (AST) to handle the problem of code synthesis. The work [15] used basic Seq2Seq model to synthesize the API usage patterns based on natural languages. The aims of language model and the decoding

module in code synthesis models are similar: to generate the most likely next code. Thus, we compare these methods in the experiments. In the problem of program translation, Tree2Tree [16] model is proposed.

II. TREE LANGUAGE MODEL

A. Preliminary

AST(Abstract Syntax Tree): Figure 1 shows an example of an expression and its corresponding AST. In this figure, the tree is placed horizontally: the root is on the left and the leaves are on the right. In rest of this paper, all trees are placed horizontally.

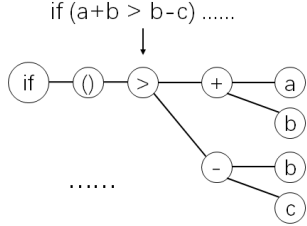


Fig. 1. an example of AST

Concepts on AST: For the abstract syntax tree of source code, some concepts need to be given.

- τ_n : the content (token) of node n is τ_n
- Sibling nodes: If two nodes have the same parent, they are siblings. For example, in Figure 1, node a and node b are sibling nodes, node $+$ and node $-$ are sibling nodes.
- Previous sibling node: If node m appears prior to node n and m and n are sibling nodes, m is called the previous sibling of n . For example, in Figure 1, node a is located above node b , so a is the previous sibling of b .
- Previous adjacent sibling node: If node m is the previous sibling of node n and m, n are adjacent, node m is called the previous adjacent sibling of node n . For example, in Figure 4, n is the previous adjacent sibling of o .
- Sibling trees: If roots of two sub-trees are siblings, these two sub-trees are called sibling trees. For example, in Figure 4, the sub-tree rooted at m and the sub-tree rooted at n are sibling trees, sub-tree rooted at node m and the sub-tree rooted at node o are sibling trees.
- Previous sibling tree: If the root of tree T_1 is the previous sibling of the root of tree T_2 , tree T_1 is the previous sibling tree of tree T_2 . For example, in Figure 4, the tree rooted at node m is the previous sibling tree of the tree rooted at node n .

LSTM: In this paper, the main logics of LSTM and two-dimensional LSTM (2DLSTM) will be expressed as two functions: LSTM and 2DLSTM. These two functions are shown in Figure 2. Function $\text{LSTM}(x, \text{cell}, h)$ takes three inputs while the function $\text{2DLSTM}(x, \text{cell}, h, \text{cell2}, h2)$ takes five inputs. The outputs for this two functions are same: $\text{cell}_{\text{new}}, h_{\text{new}}$. In this paper, parameter x is the embedding of node token

on abstract syntax tree of source code. In LSTM, cell, h pair is taken as accumulated information from one direction, and 2DLSTM receives accumulated information: $\text{cell}, h; \text{cell2}, h2$ from two directions. Please consult [17] for more information about LSTM and two-dimensional LSTM.

Algorithm LSTM(x, cell, h)	Algorithm 2DLSTM($x, \text{cell}, h, \text{cell2}, h2$)
Require: x, cell, h	Require: $x, \text{cell}, h, \text{cell2}, h2$
Ensure: $\text{cell}_{\text{new}}, h_{\text{new}}$	Ensure: $\text{cell}_{\text{new}}, h_{\text{new}}$
$i = \sigma(W_i \cdot [x, h] + b_i)$	$i = \sigma(W_i \cdot [x, h, h2] + b_i)$
$j = \tanh(W_j \cdot [x, h] + b_j)$	$j = \tanh(W_j \cdot [x, h, h2] + b_j)$
$f = \sigma(W_f \cdot [x, h] + b_f)$	$f1 = \sigma(W_{f1} \cdot [x, h, h2] + b_{f1})$
$o = \sigma(W_o \cdot [x, h] + b_o)$	$f2 = \sigma(W_{f2} \cdot [x, h, h2] + b_{f2})$
$\text{cell}_{\text{new}} = f * \text{cell} + i * j$	$o = \sigma(W_o \cdot [x, h, h2] + b_o)$
$h_{\text{new}} = o * \tanh(\text{cell}_{\text{new}})$	$\text{cell}_{\text{new}} = f1 * \text{cell} + f2 * \text{cell2} + i * j$
return $\text{cell}_{\text{new}}, h_{\text{new}}$	$h_{\text{new}} = o * \tanh(\text{cell}_{\text{new}})$
	return $\text{cell}_{\text{new}}, h_{\text{new}}$

Fig. 2. LSTM and two-dimensional LSTM

TreeNN: Given a tree T , TreeNN [10] generates vector representation of a tree. TreeNN is defined in Algorithm 1. In Algorithm 1, the symbols: c_1, c_2, \dots, c_k represent all k children of node n , $[\cdot, \cdot]$ is the vector concatenation operator, σ is the activation function such as \tanh .

Algorithm 1 TreeNN(node n)

Require: node n
Ensure: representation of tree rooted at node n
if n is a leaf **then**
 $h = \text{embedding of } \tau_n$
else
 $h = \sigma(W_{\tau_n} \cdot [\text{TreeNN}(c_1), \dots, \text{TreeNN}(c_k)])$
end if
return h

Additional Functions Other functions used are defined here.

- $\text{GetLastChildOfNode}(n)$: returns the last child of tree node n . If n does not have any child, function returns *null*. In Figure 1, node c is the last child of node $-$, $\text{GetLastChildOfNode}(-)$ returns c .
- $\text{GetPrevAdjacentSiblingOfNode}(n)$: returns previous adjacent sibling of node n , if n does not have a previous sibling (n is the first child of its parent), returns *null*. In Figure 1, node $+$ is the previous adjacent sibling of node $-$, then $\text{GetPrevAdjacentSiblingOfNode}(-)$ returns $+$.
- $\text{Parent}(n)$: returns the parent node of node n in AST.

B. Encoding model

Encoding model such as TreeNN generates the representation for a tree through visiting the tree in post-order traversal. Figure 3 gives an illustration of the execution flow of encoding model. We propose the novel encoding model based on two-dimensional LSTM: AccuredTreeGen model. AccuredTreeGen model is the one used in Decoding model of Tree Language Model. The aim of AccuredTreeGen(n) is to generate the representation for a set of trees: the tree T rooted at node n and all previous sibling trees of tree T . Unlike

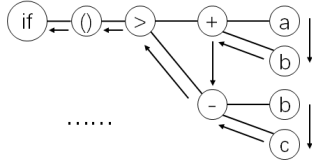


Fig. 3. data flow of encoding

traditional tree encoding model which generates encoding for a tree, The AccuredTreeGen model generates encoding for a set of trees. Figure 4 illustrates the difference of trees

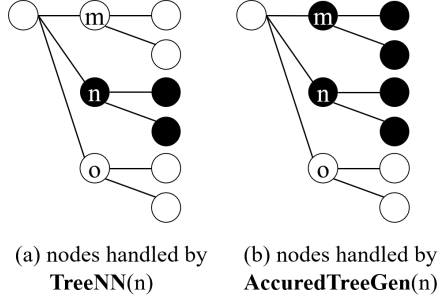


Fig. 4. An illustration of TreeNN and AccuredTreeGen

handled by TreeNN(n) and AccuredTreeGen(n). The black circles are nodes in trees to be processed. TreeNN(n) only handles the tree rooted at node n while AccuredTreeGen(n) extraly handles all previous sibling trees of the tree handled by TreeNN(n).

Algorithm 2 AccuredTreeGen(node n)

Require: node n
Ensure: representation of tree with previous sibling trees
if $n == null$ **then**
 return $cell_{zero}, h_{zero}$
end if
 $embed_{\tau_n} = \text{embedding of } \tau_n$
 $sibling_{prev} = \text{GetPrevAdjacentSiblingOfNode}(n)$
 $child_{last} = \text{GetLastChildOfNode}(n)$
 $cell, h = \text{AccuredTreeGen}(sibling_{prev})$
 $cell2, h2 = \text{AccuredTreeGen}(child_{last})$
 $cell_{new}, h_{new} = 2DLSTM(embed_{\tau_n}, cell, h, cell2, h2)$
return $cell_{new}, h_{new}$

The algorithm of AccuredTreeGen is defined in Algorithm 2. AccuredTreeGen is a recursive model. For a node n , apart from the embedding of node n , the result of AccuredTreeGen(n) depends on the result of AccuredTreeGen(previous adjacent sibling of n) and the result of AccuredTreeGen(last child of n). Recursively, For the last child: $child_{last}$ of node n , the result of AccuredTreeGen($child_{last}$) depends on the result of AccuredTreeGen(previous adjacent sibling of $child_{last}$) and the result of AccuredTreeGen(last child of $child_{last}$). Keep computing the dependency recursively, we will find

that the result of AccuredTreeGen(n) is the accumulated information of all nodes on a set of trees: tree T rooted at node n and all previous sibling trees of tree T . If node n is null, AccuredTreeGen(n) will return $cell_{zero}$ and h_{zero} which are fixed default zero values. This is also the termination for the recursive AccuredTreeGen model. The gates used in two-dimensional LSTM make the model less troubled by vanishing gradient problem than TreeNN.

C. Decoding model

Decoding model traverses from the root to the leaves on a tree in pre-order to predict the token of each node. Figure

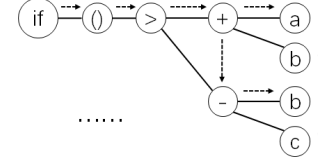


Fig. 5. data flow of decoding

5 illustrates the execution flow of the Decoding model. The Decoding model contains two sub-models: DecodeFirstChild model and DecodeNextSibling model. When we are visiting node n in pre-order traversal, DecodeFirstChild(n) generates prediction information for predicting the first child of node n . DecodeNextSibling(n) generates prediction information for predicting the next sibling of node n . The information generated by DecodeFirstChild or DecodeNextSibling consists of two vectors ($cell, h$). Every node except the root is either the first child or the next sibling of some node. So each node except the root can receive the prediction information from its parent or its previous adjacent sibling. The root node receives the fixed default values. The function FetchPrediction(node n) is defined to get the prediction information generated for predicting node n . The definition is in Algorithm 3.

Algorithm 3 FetchPrediction(node n)

Require: node n
Ensure: ($cell, h$) for predicting node n
if n is root of AST **then**
 return $cell_{zero}, h_{zero}$
end if
 $parent_n = \text{Parent}(n)$
if n is the first child of $parent_n$ **then**
 $cell, h = \text{DecodeFirstChild}(parent_n)$
else
 $sibling_{prev} = \text{GetPrevAdjacentSiblingOfNode}(n)$
 $cell, h = \text{DecodeNextSibling}(sibling_{prev})$
end if
return $cell, h$

In Algorithm 3, if node n is the root of AST which means node n does not have parent or previous siblings, the default zero values: $cell_{zero}$ and h_{zero} are returned. If node n is the first child of node $parent_n$, DecodeFirstChild($parent_n$)

generates prediction information ($cell, h$) for predicting the content of node n , as shown in the *then* branch of the if-statement in Algorithm 3. If node n is not the first child of parent $parent_n$, in this case, node n must have previous adjacent sibling: $sibling_{prev}$, DecodeNextSibling($sibling_{prev}$) generates prediction information ($cell, h$) for n , as shown in the *else* branch of the if-statement in Algorithm 3. In summary, FetchPrediction(n) just fetches prediction information from the parent of n or the previous sibling of n according to the position of n in AST. Assume that n is the first child of $parent_n$, then, the information ($cell, h$) returned by FetchPrediction(n) is just the information ($cell, h$) returned by DecodeFirstChild($parent_n$). Note that the prediction ($cell, h$) for node n returned by FetchPrediction(n) can be taken as the accumulated information of nodes visited before n .

The algorithm of DecodeFirstChild model is defined in algorithm 4. The embedding of node n and the accumulated

Algorithm 4 DecodeFirstChild(node n)

Require: node n

Ensure: ($cell_{out}, h_{out}$) for first child of node n

$cell, h = \text{FetchPrediction}(n)$
 $embed_{\tau_n} = \text{embedding of } \tau_n$
 $cell_{out}, h_{out} = \text{LSTM}(embed_{\tau_n}, cell, h)$
return $cell_{out}, h_{out}$

information of nodes visited before n ($cell, h$ returned by FetchPrediction(n)) are fed into LSTM to predict the first child of node n . The FetchPrediction, DecodeFirstChild and DecodeNextSibling (described in the following) functions call each other and form a recursive neural model. To predict the sibling of a node n , the algorithm of DecodeNextSibling model is in Algorithm 5. If $child_{last}$ is the last child of node n , AccuredTreeGen($child_{last}$) generates the representation ($cell2, h2$) for a set of trees: tree $T_{lastchild}$ rooted at node $child_{last}$ and all previous sibling trees of tree $T_{lastchild}$. The nodes in tree $T_{lastchild}$ and all previous sibling trees of $T_{lastchild}$ constitute all the descendants of node n . The embedding of node n , the accumulated information of nodes visited before n ($cell, h$ returned by FetchPrediction(n)) and the representation ($cell2, h2$) for all descendants of node n are fed into two-dimensional LSTM to predict next sibling of node n . For a node n , DecodeForFirstChild(n)

Algorithm 5 DecodeNextSibling(node n)

Require: node n

Ensure: ($cell_{out}, h_{out}$) for next sibling of node n

$cell, h = \text{FetchPrediction}(n)$
 $child_{last} = \text{GetLastChildOfNode}(n)$
 $cell2, h2 = \text{AccuredTreeGen}(child_{last})$
 $embed_{\tau_n} = \text{embedding of } \tau_n$
 $cell_{out}, h_{out} = \text{2DLSTM}(embed_{\tau_n}, cell, h, cell2, h2)$
return $cell_{out}, h_{out}$

and DecodeForNextSibling(n) both use the embedding of node n . The difference between DecodeForFirstChild(n) and

DecodeForNextSibling(n) is whether or not to take all descendants of node n into consideration. When predicting the first child of node n , we do not need to take descendants of node n into consideration because no descendant of node n has been visited in pre-order traversal of AST. When predicting the sibling of node n , all descendants of this node have been visited (predicted) and we use AccuredTreeGen to explicitly encode the already visited (predicted) sub-trees. Given a tree, starting with the root of that tree, all nodes can be predicted by keeping inferring the first child of a node and the next sibling of a node. Figure 6 gives an illustration about the data flow of encoding model and decoding model to show how decoding model interacts with encoding model. In Figure 6, solid arrow means the data flow of encoding while dotted arrow means the data flow of decoding. In Figure 6, we use the content of a node to refer to that node. As shown in the figure, the nodes $()$, $>$ and $+$ are the first child of their parents. DecodeFirstChild model is used to generate prediction information for those nodes. To predict node $-$ which is the next sibling of node $+$, the embedding of token $+$, the prediction for node $+$ (the data flow is marked with dotted arrow), the representation for all descendants of node $+$ (the data flow is marked with solid arrow) will be fed into two-dimensional LSTM to generate a new $cell, h$.

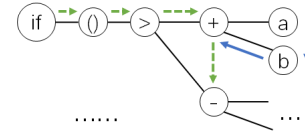


Fig. 6. an example of decoding combined with encoding

Tree Language Model is also a generalized framework in which the encoding model can be replaced with existing tree classification models. Take TreeNN as an example. If we want to use TreeNN in Tree Language Model, DecodeNextSibling model should be replaced with DecodeNextSiblingUsingTreeNN model. The definition of DecodeNextSiblingUsing

Algorithm 6 DecodeNextSiblingUsingTreeNN(node n)

Require: node n

Ensure: ($cell_{out}, h_{out}$) for next sibling of node n

$cell, h = \text{FetchPrediction}(n)$
 $encoding_{tree} = \text{TreeNN}(n)$
 $cell_{out}, h_{out} = \text{LSTM}(encoding_{tree}, cell, h)$
return $cell_{out}, h_{out}$

ingTreeNN is in Algorithm 6. The difference between DecodeNextSibling and DecodeNextSiblingUsingTreeNN is that DecodeNextSiblingUsingTreeNN uses TreeNN to encode the root node n and all its descendants into one vector instead of taking apart them. LSTM instead of two-dimensional LSTM is applied. It is a pioneering work to adapt the tree classification model as a language model. Other tree models such as TBCNN and EQNET can be adapted in a similar way.

D. Predicting and Training

For every node n , the prediction result ($cell \in \mathbf{R}^d$, $h \in \mathbf{R}^d$) for node n returned by $\text{FetchPrediction}(n)$ is used to compute the probability distribution of all tokens. The Algorithm 7 is the definition of function Predict . In Algorithm 7, $W_1 \in \mathbf{R}^{t \times d}$ and $bias \in \mathbf{R}^t$ are model parameters, t represents the total number of unique tokens in data set, d is the length of the embedding vector for one token. The $probs$ is the probability distribution for all tokens. The top k elements with the highest probabilities will be the final complement result. Top- k accuracy is computed in the way that if the desired token appears in the top k recommended tokens, the prediction is right, otherwise, the prediction is wrong.

Algorithm 7 $\text{Predict}(\text{node } n)$

```

 $cell, h = \text{FetchPrediction}(n)$ 
 $logits = \tanh(W_1 \cdot h + bias)$ 
 $probs = \text{softmax}(logits)$ 
return  $probs$ 

```

For every node n , training is to maximize the probability of n . This is achieved by minimize the loss of node n which is computed by function ComputeLoss . The Algorithm 8 is the definition of function ComputeLoss . In Algorithm 8, the $probs$ is the probability distribution for all tokens returned by $\text{Predict}(n)$. The $probs[\tau_n]$ means choosing the probability of τ_n (the actual content of node n) from $probs$. The final

Algorithm 8 $\text{ComputeLoss}(\text{node } n)$

```

 $probs = \text{Predict}(n)$ 
 $loss = -\log(probs[\tau_n])$ 
return  $loss$ 

```

loss is the summation of loss computed by ComputeLoss for each node in each AST. The training of the whole model is to minimize the final loss.

III. IMPLEMENTATION

Source code [18] of all models along with all data sets has been public. The source code is parsed into AST through Eclipse JDT. The implementation is based on Deep learning platform: TensorFlow. The model parameters consists of parameters in LSTM, parameters in 2DLSTM and W_1, W_2 in Algorithm 7. Because TensorFlow does not offer the implementation of two-dimensional LSTM, the logic of two-dimensional LSTM is implemented by ourselves. The physical environment is the computer with Windows 10 64 bit OS, Intel i7-6850k CPU, 32G memory and one Geforce GTX 1080 Ti GPU. The learning rate and the momentum are automatically decided by Adam optimizer in TensorFlow. Global norm is used to clip the gradient. Examples are trained or tested one by one. The representation size (alias as embedding size or feature size) for one token is 128. We will keep training the model until the prediction accuracy on the validation set does not exceed the optimal value for 50 epochs.

IV. EXPERIMENT

Without loss of generality, one of the most widely used programming language Java is chosen to conduct experiments. Java projects with high number of stars on GitHub are extracted and filtered into different data sets. Source code in each data set is divided into training set, validation set and test set in the proportions 60%, 15%, 25%. Every function declared in Java files will be parsed into an abstract syntax tree and every tree node in AST will be predicted to compute the prediction accuracy. The tree will be flattened into a sequence to apply sequential models such as LSTM. The sequence is generated by traversing the tree in pre-order and appending the encountered node back to the sequence. We mark the 0.15% least frequently occurred code tokens in training set and all unseen tokens in validation set or test set as *UNK*.

Date Sets: Three data sets: Dataset A, Dataset B, Dataset C are collected to conduct experiments to examine the performance of models. Details are shown in the following Table. Dataset A consists of all java files in project *apache commons*

	From Projects	Size	Vocabulary
Dataset A	apache commons io	2.0MB	5807
Dataset B	google guava	7.7MB	7538
Dataset C	Activiti & ESPlorer & AbFab3D & JComicDownloader	8.2MB	39886

io on Github. The project *google guava* is a Java project marked with 26554 stars on Github. Dataset B consists of Java files whose size is larger than 8K Bytes in the main module of *google guava*. Dataset C are Java files whose size is larger than 40K bytes in projects *Activiti* (3909 stars), *JComicDownloader* (188 stars), *ESPlorer* (733 stars), and *AbFab3D* (59 stars). The evaluation results on Dataset C may truly reflect the ability of each model because abstract syntax trees from Dataset C are huge. The fourth column headed by *Vocabulary* in Table IV means the quantity of unique tokens (the content of node on AST) on the data set.

Evaluation: In this section, Tree Language Model using the newly proposed encoding model: AccuredTreeGen model (based on two-dimensional LSTM) is abbreviated into TLM. Tree language model using TreeNN as the encoding model is abbreviated into TLM-TNN. LSTM and the decoding module in Tree2Tree [16] are also included in the baselines. The top- k accuracy (value is in percentage, % is omitted to save the space) is computed on every node in tree and the final top- k accuracy is the average of top- k accuracy of all nodes in all trees. When predicting next node, the model computes the probabilities for all candidate tokens. If we rank all tokens according to probabilities from large to small. For example, the token with the highest probability ranks 1, the token with the second highest probability ranks 2. The value in column headed with *mrr* means the average of the reciprocal of the rank for each token in a data set. This metric indicts the overall prediction performance of the model. The larger the *mrr*, the better the model. The column headed with *enpy* shows the entropy (\log_2 value of the perplexity). The smaller the entropy, the better the model.

Table I is the evaluation result of different models on test set. On small data sets: DS1, TLM achieves 22.2%, 9.3% higher top-1 prediction accuracy than tree decoding module in Tree2Tree and LSTM. The performance of TLM and TLM-TNN is similar, for top-1 accuracy and mrr, TLM performs better. On large data set: DS2, Tree2Tree performs much worse than other three models. TLM performs the best in all measurements. As can be shown, if the models used for code synthesis are directly applied to code completion tasks, the performance is bad. The reason will be described later. On large data set DS3, TLM achieves 40.3%, 13.8% higher top-1 accuracy than tree decoding module in Tree2Tree and LSTM. The performance of TLM and TLM-TNN is similar, for top-1 accuracy, TLM performs better. For top-k accuracy, as k becomes larger, performances of all models tend to be closer to each other. For top-10 accuracy, performances of all models are close. This indicates that for the top-k accuracy, the smaller the k, the more effective is the top-k accuracy to illustrate performances of different models. As can be seen, both TLM and TLM-TNN performs better than the rest non-TLM models. This demonstrates the advantages of the newly proposed Tree Language Model framework which uses tree encodings to encode the context to help predict code. In future work, we may adopt other measurements to investigate different models. By considering all conditions, on all 3 data sets, Tree Language Model framework performs better than other models. TLM performs the best on top-1 accuracy.

TABLE I
PREDICTION ACCURACY ON TEST SET

		top-1	top-3	top-6	top-10	mrr	enpy
A	Tree2Tree	38.6	57.4	63.7	67.6	0.49	4.3
	TLM-TNN	46.1	61.4	66.5	69.3	0.54	4.0
	LSTM	43.2	57.5	63.7	66.7	0.51	4.2
	TLM	47.2	61.4	66.3	68.9	0.55	4.1
B		top-1	top-3	top-6	top-10	mrr	enpy
	Tree2Tree	39.3	60.8	70.3	75.6	0.52	3.5
	TLM-TNN	68.6	81.3	85.8	87.9	0.76	2.6
	LSTM	70.1	80.7	84.6.8	86.6	0.76	3.0
C		top-1	top-3	top-6	top-10	mrr	enpy
	Tree2Tree	34.7	54.0	61.7	66.1	0.46	6.2
	TLM-TNN	46.2	60.5	65.9	68.0	0.53	5.8
	LSTM	42.9	57.9	63.8	67.1	0.51	5.9
		top-1	top-3	top-6	top-10	mrr	enpy
	Tree2Tree	34.7	54.0	61.7	66.1	0.46	6.2
	TLM-TNN	46.2	60.5	65.9	68.0	0.53	5.8
	LSTM	42.9	57.9	63.8	67.1	0.51	5.9

The reason for better performance of Tree Language Model framework is adopting tree encoding model to capture the characteristics of encountered trees or sub-trees. In Tree2Tree, a tree must be converted to a binary tree. This step introduces extra nodes. The extra introduced nodes make predicting more difficult. When standing at a node, two LSTM models are used for predicting the left child node and right child node of that node separately at the same time. Predicting left without using information of right (vice versa) causes the performance declining. Current synthesis models pay much attention to predicting code based on user-provided information but pay little attention to predicting next code based on the already decoded code (context). This paper investigates this problem.

V. CONCLUSION

This paper proposed a novel tree language model consisting of decoding model and encoding model. Two-dimensional LSTM is adopted to deal with the structural characteristics of trees. The experiments demonstrate that tree language model (TLM) achieves better top-1 prediction accuracy on large data set compared to state-of-art models.

REFERENCES

- [1] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. T. Devanbu, "On the naturalness of software," in *ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, 2012, pp. 837-847. [Online]. Available: <http://dx.doi.org/10.1109/ICSE.2012.6227135>
- [2] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "A statistical semantic language model for source code," in *ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, 2013, pp. 532-542. [Online]. Available: <http://doi.acm.org/10.1145/2594291.259458>
- [3] V. Raychev, M. T. Vechev, and E. Yahav, "Code completion with statistical language models," in *PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, 2014, p. 44. [Online]. Available: <http://doi.acm.org/10.1145/2594291.2594321>
- [4] Z. Tu, Z. Su, and P. Devanbu, "On the localness of software," in *The ACM Sigsoft International Symposium*, 2014, pp. 269-280.
- [5] A. T. Nguyen and T. N. Nguyen, "Graph-based statistical language model for code," in *ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, 2015, pp. 858-868. [Online]. Available: <http://dx.doi.org/10.1109/ICSE.2015.336>
- [6] M. White, C. Vendome, M. Linares-Vasquez, and D. Poshyvanik, "Toward deep learning software repositories," in *Ieee/acm Working Conference on Mining Software Repositories*, 2015, pp. 334-345.
- [7] V. Raychev, P. Bielik, and M. T. Vechev, "Probabilistic model for code with decision trees," in *OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, October 30 - November 4, 2016*, 2016, pp. 731-747. [Online]. Available: <http://doi.acm.org/10.1145/2983990.2984041>
- [8] H. K. Dam, T. Tran, and T. T. M. Pham, "A deep language model for software code," in *FSE 2016: Proceedings of the Foundations Software Engineering International Symposium*. [The Conference], 2016, pp. 1-4.
- [9] V. J. Hellendoorn and P. Devanbu, "Are deep neural networks the best choice for modeling source code?" in *Joint Meeting on Foundations of Software Engineering*, 2017, pp. 763-773.
- [10] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631-1642.
- [11] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 1287-1293.
- [12] V. Raychev, P. Bielik, M. T. Vechev, and A. Krause, "Learning programs from noisy data," in *POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, 2016, pp. 761-774. [Online]. Available: <http://doi.acm.org/10.1145/2837614.2837671>
- [13] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Mapping language to code in programmatic context," *arXiv preprint arXiv:1808.09588*, 2018.
- [14] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," *arXiv preprint arXiv:1704.01696*, 2017.
- [15] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API learning," in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, 2016, pp. 631-642. [Online]. Available: <https://doi.org/10.1145/2950290.2950334>
- [16] X. Chen, C. Liu, and D. Song, "Tree-to-tree neural networks for program translation," *arXiv preprint arXiv:1802.03691*, 2018.
- [17] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer Berlin Heidelberg, 2012.
- [18] "The source code of models in the experiments and all data sets," <https://www.dropbox.com/s/8typta3a81htclr/TLM.zip>.

Fast Exhaustive Search Algorithm for Discovering Relevant Association Rules

Hend Amraoui*, Faouzi Mhamdi†, Mourad Elloumi‡

*†‡ Laboratory of Technologies of Information and Communication and Electrical Engineering (LaTICE),

National Higher School of Engineers of Tunis (ENSIT), University of Tunis, Tunisia

*Faculty of Science of Tunis, University of Tunis el Manar, Tunis, Tunisia

† Higher Institute of Applied Languages and Computer Science of Beja, University of Jendouba, Tunisia

*amraoui.hend@yahoo.fr, †faouzi.mhamdi@ensi.rnu.tn, ‡Mourad.Elloumi@gmail.com

Abstract—Association Rules Mining (ARM) is one of the most important tasks of Data mining. The purpose of ARM is to discover relationships having an interest between attributes/patterns stored in very large databases. Nowadays several efficient algorithms have been developed by the researchers for the discovery of relevant Association Rules (ARs). These latter are responsible for decision making in several domains, such as medicine, finance, marketing and many other fields. In this paper, we propose a new algorithm based on exhaustive search to find relevant AR to make the decision and to predict the chance of occurring the Diabetes Mellitus (DM). We develop an algorithm to mine data in less time and less complexity without losing information. Finally, we test our approach using a real database to evaluate the efficiency of our algorithm compared to Apriori algorithm.

Index Terms—Association Rules, Fast Exhaustive Search Algorithm, Support, Confidence, Fitness

I. INTRODUCTION

The task of ARM was introduced for the first time by Agrawal et al., [1] to discover relationships among attributes in databases. From these association rules, several decisions will be taken. The formal notations of ARM were introduced by Agrawal et al., [1].

Let $I = I_1, I_2, \dots, I_m$ be a set of m different attributes, T be the transaction that comprises a set of items such that $T \subseteq I$, D be a database with different transactions T_s . An association rule is an insinuation in the form of $X \Rightarrow Y$, where $X, Y \subset I$ are sets of items termed itemsets, and $X \cap Y = \emptyset$. X is named antecedent, Y is called consequent. The rule means X implies Y .

ARM goes through two steps: The first one is to identify the frequent items using minimum support threshold s and the second one is to generate the relevant rules from these frequent items using minimum confidence threshold c . Since the database is large and users concern about only those frequently purchased items, usually thresholds of support and confidence are predefined by users to drop those rules that are not so interesting or useful. The two thresholds can be specified by the users [2]. Support of an association rule is defined as the percentage/fraction of records that contain $X \cap Y$ to the total number of records in the database [2].

$$\text{Support}(XY) = \frac{\text{Support count of } XY}{\text{Total number of transaction in } D} \quad (1)$$

Confidence of an association rule is defined as the percentage/fraction of the number of transactions that contain $X \cap Y$ to the total number of records that contain X , where if the percentage exceeds the threshold of confidence, an interesting association rule $X \Rightarrow Y$ can be generated [2].

$$\text{Confidence}(X|Y) = \frac{\text{Support}(XY)}{\text{Support}(X)} \quad (2)$$

The generation of all the frequent elements is a time consuming task when the number of items is large [3]. For this purpose we developed an algorithm generating association rules in a direct way by maximizing the support/confidence of the rules skipping the frequent itemset generation step. Our approach is then to treat the problem of ARM as a multi-objective optimization problem where the goal is to find ARs while optimizing a fitness function (described below) i.e., it is a function for evaluating rules [4], it shows if an association rule is a good solution or not.

The remainder of the paper is as follows : Section II introduces some previous work on ARM, section III explains problem and technical solutions, section IV describes the proposed algorithm, the different improvements and its performance, in section V experimental results are presented and finally, the paper is concluded in Section VI.

II. LITERATURE REVIEW

A large number of previous studies have been focused on the problem of ARM. Agrawal et al., [5] developed Apriori algorithm. It first generates the set of elements from the previous element sets, then, a pruning step was applied on these candidates. Finally, ARs are deduced from the frequent candidates generated in the first step. Apriori algorithm is easy to execute and very simple but suffers from inefficiencies: scanning the database frequently; generating a large number of candidate sets [6] and demanding huge data storage [7]. Zaki [8] developed Eclat algorithm for discovering the set of frequent attributes. Zaki represented the database with a vertical representation to avoid the repeated traverse of the database. He decomposed the original search space into smaller pieces which can be processed independently in main-memory, and proposed three new search strategies for enumerating the frequent itemsets such as bottom-up search, top-down search, and hybrid search.

Han et al., [9] introduced Frequent Pattern Growth (FP-Growth) algorithm. Han et al., used three techniques: The first one was to compress the base as much as possible developing an FP-tree to avoid scanning it several times, the second one was to adopt a pattern fragment growth method to avoid the costly generation of a large number of candidate sets, and finally used divide-and-conquer method to decompose the mining task and reduce the search space.

Uno et al., [10] proposed Linear time Closed item set Miner (LCM) algorithm. They constructed tree-shaped transversal routes composed of only frequent closed item sets. LCM algorithm founded all frequent closed item sets in polynomial time per item set, without storing previously obtained closed item sets in memory.

Yuan [6] proposed an improved Apriori algorithm as a remedial to the previous defects. Yuan used a new database mapping way, pruned more candidate elements and used overlap strategy to count support. The improved Apriori algorithm achieves excellent performance by reducing the time consumed in transaction scanning for the generation of candidate itemsets and by reducing the number of transaction to be scanned.

Sheng et al., [11] proposed a novel method by combining the Apriori algorithm and probabilistic graphical model. Sheng et al. avoided the disadvantage that whenever the frequent items are searched the whole data items have to be scanned cyclically. The effectiveness and feasibility of the proposed method in ARM has been proved.

Liu et al. [12] proposed a fast Apriori algorithm, called ECTPPI-Apriori, for processing large datasets. The algorithm uses a parallel mechanism in the ECPI tissue-like P system. The time complexity of ECTPPI-Apriori is improved to $O(t)$ compared to other parallel Apriori algorithms.

Fu et al. [13] have developed a parallelization algorithm based on the Hadoop framework and the Map Reduce model. The algorithm proved its efficiency in extracting frequent elements and generating ARs from large transactional databases.

III. MATERIAL AND METHODS

A. Data Collection

For experimental evaluation, a bench mark dataset has been selected as input data <https://www.kaggle.com/uciml/pima-indians-diabetes-database>. The studied population is PIMA Indian population near Phoenix, Arizona, it is known to be one of the communities with the highest percentage of diabetes in the world. Diabetes is a multifactorial disease where multiple rare genetic, environmental and even behavioral variants influence collectively the expression and the prevalence of traits and diseases [14]. That population has been under continuous study since 1965 by the National Institute of Diabetes and Digestive and Kidney Diseases [15]. The dataset is composed of 768 patients, and two classes. The variables are medical measurements of the patient plus age and pregnancy information. The classes are : *C0*, indicates *True Diabetic Test* (268) and *C1*, indicates *False Diabetic Test* (500) [16].

Among the factors, Diabetes Pedigree Function (*DPF*) was developed by Smith et al. [15], This function takes into consideration genetic factors inherited from ascendants.

$$DPF = \frac{\sum_i K_i(88 - ADM_i) + 20}{\sum_j K_j(ALC_j - 14) + 50} \quad (3)$$

- 1) i ranges over all relatives, who had developed diabetes by the subject's examination date;
- 2) j ranges over all relatives, who had not developed diabetes by the subject's examination date;
- 3) K_x is the percent of genes shared by the relative;
- 4) ADM_i is the age in years of relative, when diabetes was diagnosed;
- 5) ALC_j is the age in years of relative j at the last non-diabetic examination (prior to the subject's examination date);
- 6) The constants 88 and 14 represent, the maximum and minimum ages at which relatives of the subjects in this study have developed diabetes;
- 7) The constants 20 and 50 were chosen such that:
 - A subject with no relatives would have a *DPF* value slightly lower than average.
 - The *DPF* value would decrease relatively slowly as young relatives free of diabetes joined the database.
 - The *DPF* value would increase relatively quickly as known relatives developed diabetes.

Table I illustrates an excerpt of data representing medical measures and observations collected after studying a set of (768) patients. The first column contains an ordering number for the patients. The eight next columns represent measures for medical factor:

- Number of times pregnant : p ,
- Plasma Glucose Concentration at 2 Hours in an Oral Glucose Tolerance Test : g ,
- Diastolic Blood Pressure (mm Hg) : b ,
- Triceps Skin Fold Thickness (mm) : s ,
- 2-Hour Serum Insulin (U/h/ml) : i ,
- Body Mass Index ((Weight in kg)/(Weight in m)²) : B ,
- Diabetes Pedigree Function : d ,
- Age (years) : a .

The last column describes the fact whether the corresponding patient is actually sick (*C1*) or not (*C0*).

TABLE I: Medical Measures

	p	g	b	s	i	B	d	a	Out
1	0	100	70	26	50	30.8	0.597	21	<i>C0</i>
2	0	100	88	60	110	46.8	0.962	31	<i>C0</i>
3	0	101	62	0	0	21.9	0.336	25	<i>C0</i>
...
768	9	184	85	15	0	30	1.213	49	<i>C1</i>

B. Discretization of Numeric Attributes (Factors)

In our database, the range of each numeric attribute is very wide. To overcome this problem it was decided to turn numeric attributes into discrete ones. Table II illustrates an excerpt for

different distinguished ranges for each measured criterion. For example, measures for the criterion *Age* (last line) may range over $\{21, 24\}$ or $\{25, 30\}$ (in fact also over $\{31, 40\}$, $\{41, 55\}$ or $\{56, -\}$ which are not represented in the excerpt).

When an interval ends with $-$, then it admits no upper bound. An interval suffixed by $*$ is the one to be picked up by default when the measured value does not fit into any interval.

The database in Table I can then be discretized by replacing each measured value for a given criterion by the identifier of the interval it fits into. The result is illustrated in Table III.

TABLE II: Medical Measures Ranges

C	R1	R2	...	R6
p	{0, 2}	{3, 6}	...	
g	{0, 89.1}	{89.2, 107.1}	...	{165.2, -}
b	{0, 0}*	{1, 76.1}	...	{98.2, -}
...
a	{21, 24}*	{25, 30}	...	

We note that the factor 10 has been chosen to obtain distinct interval ids. This is true in our example because for all factor, the number of intervals never exceeds 9. In the general case, this factor can be chosen as the closest power of 10 to the maximum number of intervals for factor to ensure interval's id uniqueness.

Uniqueness is needed here to obtain different items representing different factor ranges. This may be useful when considering sets of such items in the algorithms we present in this paper.

TABLE III: Discretized Medical Measures

	p	g	b	s	i	B	d	a	Out
1	11	22	32	43	52	63	73	81	91
2	11	22	33	44	52	65	74	83	91
3	11	22	32	41	51	61	72	82	91
...
768	13	26	33	42	51	63	75	84	92

C. Problem Description

Given the representation in Table III of the patients' database it is interesting to analyze which combination of factor ranges implies the most sickness (or no sickness). We may consider combination taking into account all or only subsets of the factor. To represent a given combination of factor ranges and the associated desired outcome (sickness or not) we are using an *Association Rule*, defined below.

definition 1: Given A a set of integers and c an integer, we define the corresponding *Association Rule* (or simply *Rule*), γ , denoted by $A \Rightarrow c$. A is called the *antecedent* of γ (and denoted by $\mathcal{A}(\gamma)$) and c its *consequence* (and denoted by $\mathcal{C}(\gamma)$). The *size* of γ (denoted by $\mathcal{S}(\gamma)$) is the size of its antecedent: $|\mathcal{A}(\gamma)|$.

An association rule $\mathcal{A}(\gamma)$ represents the logical implication: "if items in A are all observed on medical measures for a patient then the outcome for that patient is c ".

definition 2: Given an association rule γ , a projection of γ is a rule γ' , such that $\mathcal{A}(\gamma')$ is a non-empty subset of $\mathcal{A}(\gamma)$. When γ' is a projection of γ we also say γ is an enrichment of γ' .

We note that the encoded database of Table III can be represented by a sequence (array or list) of association rules. Given definitions 1,2 we can now state the problem we propose to investigate.

problem 1: Given:

- a sequence of association rules $\mathcal{P} = [A_p \Rightarrow c_p]_{p \in I}$, with the same size d , called *population*,
- an objective function f mapping each projection of an association rule in \mathcal{P} to a real value in $[0, 1]$,
- and an integer s between 1 and d .

Compute all the projections of size s of associations rules in \mathcal{P} that maximizes the objective function f .

We propose to solve the Problem 1 for all possible sizes of association rules. As objective function we will use the so-called *fitness*, defined below.

definition 3: Given two strictly positive real values α and β , and a set \mathcal{P} of association rules, the *fitness* function (denoted *fit*) maps each projection γ of elements in \mathcal{P} to a positive real value $fit(\gamma)$ as described below:

$$fit(\gamma) = \frac{\alpha \times supp(\gamma) + \beta \times conf(\gamma)}{\alpha + \beta} \quad (4)$$

where :

- the *support* of γ denoted by $supp(\gamma)$ is $\frac{y(\gamma)}{\mathcal{P}}$,
- the *confidence* of γ denoted by $conf(\gamma)$ is $\frac{y(\gamma)}{x(\gamma)}$,
- $x(\gamma)$ is the number of enrichments of γ in \mathcal{P} ,
- and $y(\gamma)$ is the number of enrichments of γ in \mathcal{P} , having the same consequence as γ .

We note that for a given association rule γ , $supp(\gamma)$ measures the probability of occurrence of items in $\mathcal{A}(\gamma)$ together with consequence $\mathcal{C}(\gamma)$ in the population, while $conf(\gamma)$ measures the probability of occurrence of the consequence $\mathcal{C}(\gamma)$ having all items in $\mathcal{A}(\gamma)$ appearing in the rules of the population.

IV. FAST EXHAUSTIVE SEARCH FOR ASSOCIATION RULE MINING ALGORITHM (FES-ARM)

To solve Problem 1 we propose the following exhaustive search algorithm. We notice that Algorithm 2 may be called by Algorithm 1 many times for the same rule. This is due to the fact that different rules can have common projections. We note that running Algorithm 2 with a rule that already have been treated does not bring more results. The proposed enhancement consists of using a *setRulesDone* which will hold all the already analyzed rules, and recursion in Algorithm 2 will only be initiated for rules that are not in *RulesDone*. There is also a second improvement: It tackles a costly part of the treatment : the computation of the fitness function itself. We propose to keep track for all computed fitnesses for all analyzed rules. This is efficient since fitness is likely to be computed many times for the same rule.

A. FES-ARM Algorithm

To solve Problem 1 we propose the following FES-ARM algorithm.

- 1) Compute all projections of rules in the population and keep only those having maximum fitness.
- 2) Projections of a given rule γ , are recursively computed by removing at each recursion step only one item from $\mathcal{A}(\gamma)$.

Algorithm 1: MAIN FES-ARM Algorithm

Input: \mathcal{P} and c
Output: $OptRules$, $OptFits$ and $RulesDone$

```

1 Initialize  $OptRules$  with  $\emptyset$  and  $OptFits$  with 0.0 for all rule sizes;
2 for  $\gamma \in \mathcal{P}$  do
3   SOLVE_FOR_RULE( $\gamma$ ,  $c$ ,  $OptRules$ ,  $OptFits$ ,  $RulesDone$ );
4 end

```

Algorithm 2: SOLVE_FOR_RULE

Input: γ , c , $OptRules$, $OptFits$ and $RulesDone$
Output: $OptRules$, $OptFits$

```

1 if  $\gamma \notin RulesDone$  then
2    $RulesDone.add(\gamma)$ ;
3    $CurrentFitness \leftarrow fit(\gamma)$ ;
4   if  $CurrentFitness > OptFits.get(\mathcal{S}(\gamma))$  then
5      $OptFits.put(\mathcal{S}(\gamma), CurrentFitness)$ ;
6      $OptRules.put(\mathcal{S}(\gamma), \emptyset)$ ;
7      $OptRules.get(\mathcal{S}(\gamma)).add(\gamma)$ ;
8   end
9   else
10    if  $CurrentFitness = OptFits.get(\mathcal{S}(\gamma))$  then
11       $OptRules.get(\mathcal{S}(\gamma)).add(\gamma)$ ;
12    end
13  end
14  if  $\mathcal{S}(\gamma) > 1$  then
15    for  $item \in \mathcal{A}(\gamma)$  do
16       $\mathcal{A}(\gamma') \leftarrow \mathcal{A}(\gamma) \setminus \{item\}$ ;
17       $\mathcal{C}(\gamma') \leftarrow c$ ;
18      SOLVE_FOR_RULE( $\gamma'$ ,  $c$ ,  $OptRules$ ,  $OptFits$ ,  $RulesDone$ );
19    end
20  end
21 end

```

Algorithm 1 takes as input the population \mathcal{P} , the considered consequence c and returns a mapping $OptRules : i \rightarrow \Gamma_i$ associating to each possible rule size i the set of rules realizing maximum fitness and a mapping $OptFits : i \rightarrow f_i$ associating to each possible rule size the actual maximum fitness. It uses a set $RulesDone$ that will contain all analyzed rules (for optimization purpose).

We note that Algorithm 1, solves several instances of Problem 1 at a time, i.e., for all rules sizes.

B. Complexity

Complexity of Algorithm 1 (using Algorithm 2) will obviously depend on concrete used implementations of mappings and sets data structure.

We propose to use Java *HashMap* (resp. *HashSet*) to implement mappings (resp. sets). These classes offer constant-time complexity for regular operations: add, remove and membership test (\in).

This means that all instructions in Algorithm 2 (the recursive call being omitted) are constant-time complexity, except for line 3 where a fitness is computed. Computing the fitness of a rule γ means running through the hole population of rules and testing whether $\mathcal{A}(\gamma)$ is a subset of the antecedent of the population member. Testing that a set A is a subset of another set B runs in $O(\min(|A|, |B|))$. This means computing the fitness of γ is $O(\mathcal{S}(\gamma) \times \overline{\mathcal{P}})$.

Algorithm 2 will be called for all projections of all rules in the initial population \mathcal{P} . There may be redundancy in those generated projections, since two rules can share projections. We note that the if test in line 1 of Algorithm 2 will be executed for all generated projections, with redundancy, while all the other instruction of this algorithm will be executed only once for each distinct generated projection. Each call to Algorithm 2 with rule γ and where the if test in line 1 will be positive, will cost $O(\mathcal{S}(\gamma))$.

Let N_{Γ}^s be the number of all computed projections of size s with redundancy and $N_{\Gamma,d}^s$ the number of all the latter projections without redundancy. We can safely infer that complexity of Algorithm 1 is

$$O\left(\sum_{s \in [1,d]} N_{\Gamma}^s + n \times \sum_{s \in [1,d]} N_{\Gamma,d}^s \times s\right)$$

We recall that d is the maximal rule size (in our example it is 8) and we note that $N_{\Gamma,d}^s$ and N_{Γ}^s are likely to be exponential in d and $\overline{\mathcal{P}}$ in the worst case.

C. Fitness Computation Enhancement

Computing the fitness for a given rule is a costly task. Indeed we explained in previous sections that it has $O(s \times n)$ complexity, where s is the rule's size and n the population size. We propose to reduce the fitness computation complexity using the following enhancements:

- 1) Provided that we already have computed a mapping $TransactionsByItems$, that sends every item $item$ to the set of the transactions (rules in \mathcal{P}) $\Gamma(item)$ where this item appears. We can compute then compute:

$$x(\gamma) = \bigcap_{item \in \mathcal{A}(\gamma)} TransactionsByItems(item) \quad (5)$$

If we extend the definition of *TransactionsByItems* to the consequences, we can also compute, for a given sequence c :

$$xy(\gamma) = \bigcap_{item \in \mathcal{A}(\gamma) \cup \{c\}} TransactionsByItems(item) \quad (6)$$

Since the complexity of computing the intersection of two sets A and B runs in $O(\min(\overline{A}, \overline{B}))$. We can infer that the complexity of computing every intersection is $O(s \times \min_{item \in \mathcal{A}(\gamma) \cup \{c\}} (TransactionsByItems(item)))$ and that reduces drastically the fitness computation (time) complexity. We note that pre-computing the mapping *TransactionsByItems* does not introduce too much cost, since it can be done while parsing rules from the database file.

- 2) We can also keep track of all computed fitnesses so far, using a mapping *ComputedFitnesses* that sends each already analyzed rule to its fitness value.

V. EXPERIMENTAL RESULTS

We propose in this section to discuss results obtained when applying the algorithm presented in this paper compared to Apriori Algorithm. We used databases containing 768, 7680 and 76800 entries, with antecedents of size 8 and 2 consequences, and executed 10 times.

- N: Number of iterations.
- C: The considered consequence.
- A.E.T : Average Execution Time

All times are measured in milliseconds (*ms*) except when a different unit is explicitly specified.

TABLE IV: Comparative table of Average Execution Time (A.E.T) provided by our Algorithm and Apriori Algorithm

n	C	A.E.T Apriori	A.E.T FES-ARM
768	91	667035	2120
	92	852392	2766
7680	91	> 48h	12855
	92		13342
76800	91		145355
	92		143346

Discussion: In this work we introduce an efficient algorithm to identify relevant ARs. Our experimental evaluation shows that complexity goes from exponential complexity $O(2^n)$ to polynomial complexity $O(\sum_{s \in [1,d]} N_{\Gamma}^s + n \times \sum_{s \in [1,d]} N_{\Gamma,d}^s \times s)$ in comparison with Apriori Algorithm (worst case) and in response time (mentioned in table IV). Compared to Apriori Algorithm, our approach proved the same efficiency in quality of generated rules with maximizing the fitness function, proving that we have not lost the quality of information as well even with the increased size of the database. These results are due to three improvements. The first one consists of using a *setRulesDone* which will hold all the already analyzed rules, to avoid analyzing the same rule several times (We propose mining specific to general).

The second one is computing the fitness of each transaction by pre-computing the mapping *TransactionsByItems* while parsing rules from the database file. The third one (It takes a costly part of the treatment) consists of keeping track for all computed fitnesses for all analyzed rules. This was efficient since fitness was likely to be computed many times for the same rule.

VI. CONCLUSION

This paper presents a new Fast Exhaustive Search algorithm FES-ARM for discovering efficient ARs to predict the chance of occurring the Diabetes Mellitus (DM). The proposed algorithm produces in a correct way relevant association rules (The same rules produced by Apriori algorithm) while optimizing the average execution time in the worst case to 314,64%. FES-ARM is tested on real databases containing 768, 7680 and 76800 entries, with antecedents of size 8 and 2 consequences, and executed 10 times. The complexity of FES-ARM Algorithm is improved from $O(2^n)$ to $O(\sum_{s \in [1,d]} N_{\Gamma}^s + n \times \sum_{s \in [1,d]} N_{\Gamma,d}^s \times s)$ in comparison with Apriori Algorithm. Our ambition for the future works, dealing with large transactional databases, is to develop new metaheuristics approaches to solve association rules mining as a combinatorial optimization problem.

REFERENCES

- [1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Acm sigmod record*, vol. 22, no. 2. ACM, 1993, pp. 207–216.
- [2] Q. Zhao and S. S. Bhowmick, "Association rule mining: A survey," *Nanyang Technological University, Singapore*, 2003.
- [3] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. C. Coelho, "Survey of multiobjective evolutionary algorithms for data mining: Part ii," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 1, pp. 20–35, 2014.
- [4] H. R. Qodmanan, M. Nasiri, and B. Minaei-Bidgoli, "Multi objective association rule mining with genetic algorithm without specifying minimum support and minimum confidence," *Expert Systems with applications*, vol. 38, no. 1, pp. 288–298, 2011.
- [5] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.
- [6] X. Yuan, "An improved apriori algorithm for mining association rules," in *AIP Conference Proceedings*, vol. 1820, no. 1. AIP Publishing, 2017, p. 080005.
- [7] B. Wang, D. Chen, B. Shi, J. Zhang, Y. Duan, J. Chen, and R. Hu, "Comprehensive association rules mining of health examination data with an extended fp-growth method," *Mobile Networks and Applications*, vol. 22, no. 2, pp. 267–274, 2017.
- [8] M. J. Zaki, "Scalable algorithms for association mining," *IEEE transactions on knowledge and data engineering*, vol. 12, no. 3, pp. 372–390, 2000.
- [9] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 1–12.
- [10] T. Uno, T. Asai, Y. Uchida, and H. Arimura, "Lcm: An efficient algorithm for enumerating frequent closed item sets," in *Fimi*, vol. 90. Citeseer, 2003.
- [11] G. Sheng, H. Hou, X. Jiang, and Y. Chen, "A novel association rule mining method of big data for power transformers state parameters based on probabilistic graph model," *IEEE Transactions on Smart Grid*, vol. 9, no. 2, pp. 695–702, 2018.
- [12] X. Liu, Y. Zhao, and M. Sun, "An improved apriori algorithm based on an evolution-communication tissue-like p system with promoters and inhibitors," *Discrete Dynamics in Nature and Society*, vol. 2017, 2017.

- [13] C. Fu, X. Wang, L. Zhang, and L. Qiao, "Mining algorithm for association rules in big data based on hadoop," in *AIP Conference Proceedings*, vol. 1955, no. 1. AIP Publishing, 2018, p. 040035.
- [14] H. Amraoui, F. Mhamdi, and M. Elloumi, "Survey of metaheuristics and statistical methods for multifactorial diseases analyses," *AIMS Med Sci*, vol. 4, pp. 291–331, 2017.
- [15] J. W. Smith, J. Everhart, W. Dickson, W. Knowler, and R. Johannes, "Using the adap learning algorithm to forecast the onset of diabetes mellitus," in *Proceedings of the Annual Symposium on Computer Application in Medical Care*. American Medical Informatics Association, 1988, p. 261.
- [16] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

Collecting Data from Continuous Practices: an Infrastructure to Support Team Development

Ana Filipa Nogueira^{*†}, Emilien Sergeant[†], José C. B. Ribeiro[‡],
Mário A. Zenha-Rela^{*} and Antoine Craske[†]

^{*} CISUC, University of Coimbra, Portugal

[†] DOSI, La Redoute S.A., France

[‡] CIIC, Polytechnic Institute of Leiria, Portugal

Abstract—Through software analytics, raw data with low value originates information that is valuable and able to provide insights, enabling the support of claims that would otherwise not be possible to verify. The software development ecosystem has plenty of sources that can help understanding the quality of processes and products but, to reach that goal, it is necessary to collect and store the data. This paper describes an infrastructure to allow the collection, storage and analysis of data from software repositories. The scope of the research is an industrial case study, which encompasses several specificities: tools and work methodology. The current solution is able to collect information from the continuous delivery & deployment pipeline, which includes data sources such as the source code repository (SVN), the static analysis tool (SonarQube), the continuous integration server (from Jenkins jobs) and the continuous testing tool (an in-house tool called Cerberus). Future work also includes the implementation of components that will allow the collection of unstructured data from the bug-tracking system and incident management tool. As stated in the literature, correlating the history of issues and incidents will allow the team to address, or at least identify, areas of improvement.

Index Terms—CI/CD Pipeline, Mining Software Repositories, Machine Learning, Software Analytics, Software Quality

I. INTRODUCTION

Software Analytics as a means to predict or gather information about software development activities is an engaging research field that aims to provide insights and solutions, not only for the technical topics, but also for organizational issues. In general, one's aspiration is to guide teams to enhance the quality of the products being developed and processes being executed ([1], [2], [3], [4]). In the scope of software development, the ability to perform change impact analysis is especially relevant since team members will be able to receive direct feedback from their work and at the same time forecast or identify support for other tasks such as test-case selection and prioritization [5].

Mining Software Repositories (MSR) is among the new techniques used to perform change impact analysis [6]; this technique makes use of historical data available in software repositories, such as issue tracking systems and source code repositories. Menzies and Zimmermann write about the common vision shared in the literature [4]: “data from software

projects contains useful information that can be found by data-mining algorithms”. As such, commit messages and bug-reports are examples of artifacts from which knowledge related to change impacts can be extracted. For instance, Ying *et al.* [7] claim that a developer may benefit from patterns in the history change of a system while performing maintenance tasks. Therefore, well-known applications scope include recommendation systems, and effort and defects predictors for software development. New fields of research include: detectors for unexpected or bug-prone code, indicators for developer's mood by applying sentiment analysis to comments, and explorers for complex spaces. As for data science in the industry, what was seen in the past as cutting-edge research is now part of daily operations of IT companies [4], [8].

With the current shift concerning automated delivery and deployment, and when moving from manual and bureaucratic processes to almost full automation, all those tools pose as data sources. Nowadays, delivery and deployment are continuous practices supported by tools [9], and the capture of events is more straightforward. In addition to mine source repositories and bug tracker systems, the spectrum of sources may include: code review tools [10], Continuous Integration/Continuous Delivery (CI/CD) tools or automated tools for continuous testing. Even though there are principles and practices transversal to companies that implement continuous practices, a company's specific characteristics influence how these practices are implemented [11], including the business domain, the type of product being developed, or the impact on the customers (e.g., some companies report that frequent releases may lead to an increase on customer churn rate).

In order to gain visibility of all the events generated by continuous practices it is necessary to implement an infrastructure to support the data collection and storage. This paper reports on a real case study, implemented for the IT department of La Redoute¹, which is an *e-commerce* company. The infrastructure presented aims to collect data from several data sources available in the continuous pipeline: source code repository, static analysis tool, CI/CD server and continuous testing tool. The paper is the continuation of previous work

[12]; it described the ongoing research and motivation to apply machine learning techniques to improve the quality of processes and products developed in La Redoute's IT department. A particularity of the architecture presented in this paper is that it is emerging during the digital transformation of the company, i.e., new projects moving to micro-services cloud-based architecture, in opposition to monolithic solutions implemented in the past. The infrastructure being developed is also supported by this new technological stack which includes components such as Kafka², Kubernetes³, Elasticsearch, Logstash and Kibana (from the Elastic stack⁴). Considering some the advantages of software analytics and MSR, this infrastructure will support the team's need to anticipate issues ("Is this commit considered problematic?") and to understand the best and worst practices in place, both technical and social ("Do people on my team need training?").

The rest of paper is organised as follows: Section II overviews relevant literature; Section III describes the main requirements and the architecture being implemented, and finally, IV concludes this paper while identifying future directions.

II. RELATED WORK

This section overviews literature that takes advantage of repositories mining in order to find patterns and to extract knowledge about the aspects influencing the software development and evolution.

Ball *et al.* [13] proposed a framework to delve into the relationships between several components of the software development process: requirements, technological stack, software development and the development's team organizational characteristics. In the scope of C++ case studies, the authors derived a VCS-related metric: connection strength determined with basis on how probable it is that two classes are modified together. Additionally, Ball *et al.* highlighted the existence of valuable contextual information that could be leveraged to understand how a component evolved from one release to another; examples include code modified, date and time of modification, and the author. The assumption is that automated analysis can also be applied to contextual data.

Hipikat tool [14] was conceived by Čubranić and Murphy as a means to aid newcomers to an open-source project, which don't have the same support net when compared to traditional *in-house* teams. The authors use the concept of *implicit group memory* that infers links between archived artifacts produced in the source repository, issue-tracking systems, communication channels and online documentation. The *implicit group memory* is then used to recommend artifacts, from the archives, possibly relevant to the task assigned to the newcomer.

Zimmermann *et al.* [15] present the tool ROSE – a plugin for Eclipse IDE⁵ – that guides the programmer. An association rule mining algorithm was employed to mine the *version*

histories providing the developer means to: i) "suggest and predict likely changes"; ii) "prevent errors due to incomplete changes"; and iii) "detect coupling undetectable by program analysis". As the tool is based on the files' version history, it is possible to observe another type of coupling that is not code-based: coupling between items that are not applications or programs.

Canfora and Cerulo [16] proposed a method to infer the list of impacted source files that will be impacted by a change request. The method makes use of information from the source repository and the changes requests (Bugzilla⁶) and via information retrieval algorithms, the technique makes the link between the new change request and the historical revisions impacted by similar requests. The evaluation of the method was implemented in four open source projects, and the positive results range from 30% to 78%. The prediction of the effort required to test is another possible application pointed by the authors that will help both developers and project managers.

Ren *et al.* developed Chianti [17], a tool to support change impact analysis of Java programs, integrated in the context of Eclipse IDE. For this tool in particular, regression and unit tests, and corresponding executions, are the artefacts of interest, i.e., the tool reports on how tests' behaviour are influenced by changes. For each change on a test's behaviour, the tool also determines the "affecting changes".

Ying *et al.* [7] developed an approach that relies on data mining techniques to identify patterns on the change history of the base code; specifically, the approach searches for patterns among the changes observed on the set of files that commonly change together. Based on their approach, the authors report that history change patterns support recommendation systems that indicate additional code that should be modified when a developer is doing a modification of the source code. This work is aligned with work done by Zimmermann *et al.* [15] even if applying different algorithms.

Zanjani, Swartzendruber and Kagdi [6] also presented an approach called *InComIA* to understand the change impacts of an incoming request. They aim for connections between historical data, provided in task management applications (Mylyn⁷), and histories and contextual information available in commits (via source code repositories). A *corpus* of source code entities – methods and files – was created by applying several techniques: information retrieval, machine learning and source code analysis. For an incoming request, its text is used to query the *corpus* and as a result, the tool returns a list of the most prone to change entities.

TARMAQ is an algorithm developed by Rolfsnes *et al.* [5] for mining *evolutionary coupling* (a driver for change impact analysis); it was empirically validated on six projects – 2 industrial and 4 open source – and the authors claim better results when compared with ROSE tool [15], as it worked better for heterogeneous systems. *Evolutionary coupling* aims to understand how systems evolved during their lifecycle,

²<https://kafka.apache.org/documentation/>

³<https://kubernetes.io/>

⁴<https://www.elastic.co/>

⁵<https://www.eclipse.org/ide/>

⁶<https://www.bugzilla.org/>

⁷<https://www.eclipse.org/mylyn/>

the goal is to pick data that changed “together” and mine the connections between the entities that were modified. The authors focused at connections between files, although several granularity levels would be possible (methods or variables).

Chatley et Jones [18] presented a tool called *Diggit* which is able to generate code review comments in an automated fashion; the authors used historical changes from a Git repository, using a mining algorithm to pinpoint directions on eventual changes. The tool was integrated in a development team inside an industrial case study, and the authors also reflected on the impacts of adapting an academic research into the real world to be used by developers. The authors pinpoint potential both for commercial and open-source projects (which may include a higher number of developers and number of commits during the time). Motivation for their work encompasses not only the assurance of commits’ quality but also the support to improve developers’ competences.

Following a different perspective, Mens and Goeminne [19] studied the social component applied on the open-source community: work methodologies, cooperation, communication and information sharing. The main motivation was to understand how communities impact the evolution of a software product, some of the events that were analysed by authors are also true for industrial case studies, examples, departure of a key developer or the handover of a project to another team.

Murgia *et al.* [20] also adopted a more social approach by mining issue reports to gather emotional information about the software development. The authors focused on the issue tracking system of Apache Software Foundation⁸ and were able to observe emotions such as sadness, joy and gratitude, through a human observation of the reports. The motivation for analysing this type of information is on the fact that the lack of happiness or safety may lead developers to fall behind. Moreover, to support the importance of context in the quality, Bird *et al.* [21] report about the impacts of organizational aspects, which are seen as strong indicators of quality, refuting that geographically teams are producing worst products. Menzies and Zimmermann [4] also agree that social factors are promising quality predictors.

Another level of repositories mining is to mine repositories of repositories (RoRs). For instance, Sowe *et al.* [22] studied the types of projects being developed in the open source community, for that purpose the authors used metadata available in the RoRs FLOSSmole⁹.

Our research addresses a industrial case study, i.e., the scope include projects developed by an IT department composed by almost 110 people from Development and Operations. The underlying motivation for the implementation of the infrastructure described in this document is to have means to support and improve products, processes and competences. Even though there are a few studies reporting about industrial case studies, the majority of the aforementioned research focuses on open-source case studies. Nevertheless, those are a representative

sample of complex projects both in the technical and social perspectives.

III. ARCHITECTURE

A. Continuous Delivery & Deployment Pipeline

Figure 1 depicts a general view of the delivery & deployment pipeline implemented by the IT team; it is supported mainly by Jenkins¹⁰, integrated with the other tools so as to allow the build, static analysis, deploy and testing of the components being delivered and deployed.

The pipeline is triggered immediately upon a commit on the version control system, SVN¹¹. Jenkins listens for modifications on the code repository and initiates the build of the component after each new commit. In the same job instance, after the build, the component is submitted to SonarQube¹², which, in turn, executes the static analysis (reports about technical debt, test coverage, complexity) and enforces quality rules that should be respected. If the build and quality rules are valid, the project is deployed in a first non-production environment QA – *Quality Assurance*, so that developers can perform their first tests in a machine other than theirs. A successful deployment triggers a functional test campaign, ensuring non-regression, implemented through an *in-house* tool named Cerberus [23]. The global output of the test campaign – OK or KO – is used to decide if the component should be installed in the following environments in the pipeline. After QA, the team can move their components to the next environments – UAT and PROD. UAT or *User Acceptance Tests* is a pre-production environment allowing to perform more tests in a confined environment. After the deploy of a component in the UAT environment, the corresponding automated test campaigns will be triggered as well. For Production, it is also possible to execute automated tests.

B. Architecture

Altogether, the proposed architecture (Figure 2) collects structured data about: the commits, the Jenkins jobs status for build, deployment and test campaigns; SonarQube measurements and the summary for test campaigns executions.

As mentioned in the Introduction, the architecture described in this paper is implemented in a Kubernetes cluster, which is an “open-source system for automating deployment, scaling, and management of containerized applications”. This type of infrastructure eliminates most of the manual work required to set up and modify an infrastructure, in opposition to traditional Virtual Machines. Thence, if necessary, a new resource for data collection, treatment or analysis can be easily set up and integrated in the infrastructure.

The entry or starting point of the data collection process is the Kafka component – identified as “Data pipeline”. Kafka is a distributed streaming platform that allows, among other features, to implement the streaming of data pipelines in real-time. For this specific architecture, the jobs in the Jenkins

⁸<https://www.apache.org/>

⁹<https://flossmole.org/>

¹⁰<https://jenkins.io/>

¹¹<https://subversion.apache.org/>

¹²<https://www.sonarqube.org/>

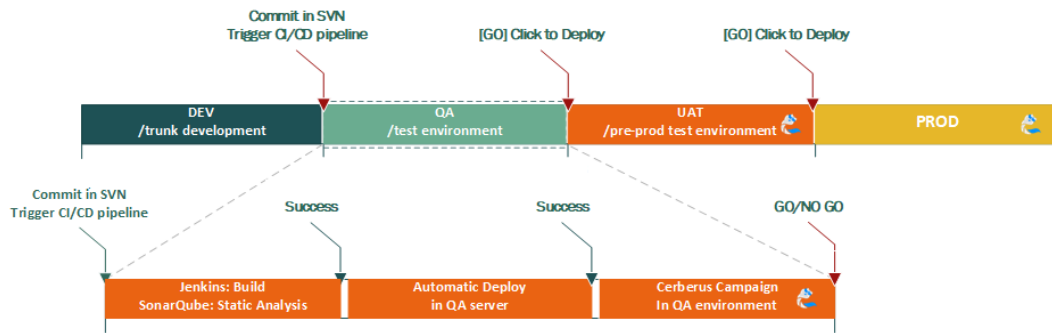


Fig. 1. Overview of development pipeline supported by Jenkins, SVN, SonarQube and Cerberus.

pipeline send data to a specific Kafka topic called “jenkins”. A *topic* defines a stream that holds specific types of events or messages. An advantage is that any consumer can pick a message or event in a topic and treat it according to their specific use case. The potential of streaming the events triggered in the continuous pipeline is huge, for instance, the “jenkins” topic is also supporting some release engineering tasks, since the team is able to extract statistics about the team’s velocity and the time necessary to reach production environments.

In this architecture, the consumer of the “jenkins” topic is referred as “Data filter”, which is a Logstash service that reads the messages and extract the meaningful data for this research. Logstash is an *Extract, Transform, and Load (ETL)* component. The “Data filter” will parse and load the useful data into a component implemented in Java, which in turn is responsible for correlating the data received with metrics from: i) the code quality analysis and ii) the result of the functional test campaign. This step will promote the construction of a complete data set that will serve the purpose of the system. The data will then be stored inside a special component, the “Data Repository”, which is an Elasticsearch engine that stores data as JSON documents. Elasticsearch allows quick searches and integrates very well with other technologies simplifying both data analysis and visualization tasks.

The second half of the architecture (“Decision Making Process”), represents the components that can consume data from the “Data repository”. It is expected to use Machine Learning techniques to predict the severity level of the events happening in the continuous pipeline. For that purpose, a first TensorFlow¹³ component is being implemented in the Kubernetes cluster. Additionally, it is important to provide means to visualize the data, and for that purpose we expect to implement useful dashboards. To begin, Kibana is also available in the cluster, facilitating the access and visualization of the data available in “Data Repository”.

With basis on this architecture, the next step is to implement all the processes and components that will analyse the data

and provide valuable insights to users, either developers or managers. For that purpose, future work will need to ensure some basic requirements [24] [18]:

- *Short execution times*: if the analysis takes too long, developers will feel tempted to abort it and to move on.
- *Short number of false positives*: a high number of false positives may lead to the abandonment of the framework.

C. Other data sources

Other data sources included in the “Data repository”, but that are currently being manually loaded include:

- *Developers experience*: the different developer categories are used as input. Developer’s data needs to be managed carefully to avoid exposing personal details. Future work may include information about the managers and team organization in order to understand the organizational impacts.
- *Impact of the project*: currently being loaded from an excel file; however, there is an ongoing project to implement a database repository which will ease the management of this type of information. Also, it is necessary to ensure that the owners of each software component share accurate information about the impacts and severity for the business.

A next step for this architecture includes the development of specific connectors to gather non-structured data from: i) Mantis – the bug tracking system for non-production issues; ii) *iTop* – system used to report issues detected in PROD environment; and iii) logs – any type of execution logs.

D. Early Observations

Even though is to soon to comment on the data collected and the insights that are possible to infer, it is interesting to report a curious observation. Linear regression was applied to two variables: i) the result of the Jenkins build job, and ii) the time of the day that a commit was done. An unexpected high number of build jobs, which are triggered by the commit, failed in the hour after the lunch break. A possible research

¹³<https://www.tensorflow.org/>

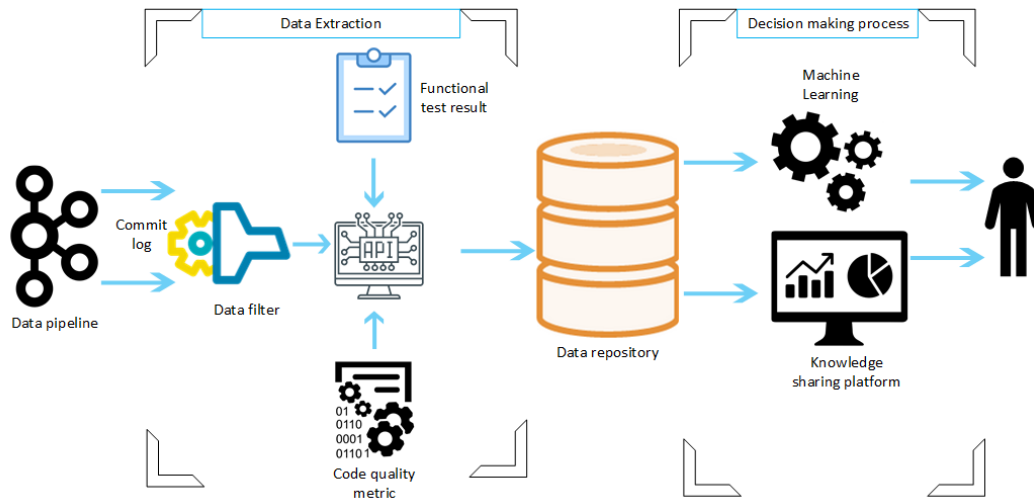


Fig. 2. Architecture to collect data from the CI/CD pipeline.

direction: What are the impacts of interrupting the “thinking flow” of a developer?

IV. CONCLUSIONS AND FUTURE DIRECTIONS

Even though software analytics opens several possibilities for “discovering, verifying, and monitoring the factors that affect software development”, there are several underlying issues [4]. The following paragraphs describe how those issues relate to the work described in this paper.

Even though the described architecture is prepared to collect data from several sources in the pipeline, there are factors that are not visible in those components nor collected in an automated manner. For that purpose, two surveys were conducted to obtain information that is not available in the tools used, but that is more related to the social component and team’s characteristics. This type of activity is time-consuming, but the surveys results are expected to uncover not only the points of improvement, but also the good practices implemented by the company.

The context may be another issue, as it may differ among projects, even when handled by the same team. Currently, we are restricting the research scope to new projects that have a high-level of readiness to integrate a micro-services architecture. Nevertheless, the legacy code, which is distributed across a panoply of projects using different technological stacks, represents a big part of the scope, and it may hinder bad practices that are propagated and need to be addressed by the team. This is a topic that needs to be monitored closely in order to understand how can we transfer the knowledge acquired from a project – that fits the standards of the new architecture – to a legacy project.

A frequent goal of this type of research, which focuses on the software development data, is to have actionable insights, i.e., how can we use the information available to take some real and concrete actions. This work is expected to provide information that can help teams reduce the overhead of manual tasks, raise automatic alerts in case of need, and also identify

potential needs in terms of training. These actionable actions will impact the continuous practices and drive business value, as an increase on the quality of processes and products is foreseen.

REFERENCES

- [1] H. Kagdi, M. L. Collard, and J. I. Maletic, “A survey and taxonomy of approaches for mining software repositories in the context of software evolution,” *J. Softw. Maint. Evol.*, vol. 19, pp. 77–131, Mar. 2007.
- [2] T. Menzies and T. Zimmermann, “Software analytics: So what?,” *IEEE Softw.*, vol. 30, pp. 31–37, July 2013.
- [3] M. A. de F. Farias, R. Novais, M. C. Júnior, L. P. da Silva Carvalho, M. Mendonça, and R. O. Spínola, “A systematic mapping study on mining software repositories,” in *Proceedings of the 31st Annual ACM Symposium on Applied Computing, SAC ’16*, (New York, NY, USA), pp. 1472–1479, ACM, 2016.
- [4] T. Menzies and T. Zimmermann, “Software analytics: Whats next?,” *IEEE Software*, vol. 35, pp. 64–70, Sep. 2018.
- [5] T. Rolfesnes, S. D. Alesio, R. Behjati, L. Moonen, and D. W. Binkley, “Generalizing the analysis of evolutionary coupling for software change impact analysis,” in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, pp. 201–212, March 2016.
- [6] M. B. Zanjani, G. Swartzendruber, and H. Kagdi, “Impact analysis of change requests on source code based on interaction and commit histories,” in *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, (New York, NY, USA), pp. 162–171, ACM, 2014.
- [7] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, “Predicting source code changes by mining change history,” *IEEE Trans. Softw. Eng.*, vol. 30, pp. 574–586, Sept. 2004.
- [8] M. Kim, T. Zimmermann, R. DeLine, and A. Begel, “The emerging role of data scientists on software development teams,” in *Proceedings of the 38th International Conference on Software Engineering, ICSE ’16*, (New York, NY, USA), pp. 96–107, ACM, 2016.
- [9] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” *IEEE Access*, vol. 5, no. Ci, pp. 3909–3943, 2017.
- [10] X. Yang, R. G. Kula, N. Yoshida, and H. Iida, “Mining the modern code review repositories,” *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR ’16*, pp. 460–463, 2016.
- [11] M. Leppnen, S. Mkinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. Mntyl, and T. Mnnist, “The highways and country roads to continuous deployment,” *IEEE Software*, vol. 32, no. 2, pp. 64–72, 2015.

- [12] A. F. Nogueira, J. C. B. Ribeiro, M. Z. Rela, and A. Craske, "Improving la redoute's CI/CD pipeline and devops processes by applying machine learning techniques," in *11th International Conference on the Quality of Information and Communications Technology, QUATIC 2018, Coimbra, Portugal, September 4-7, 2018*, pp. 282–286, 2018.
- [13] T. Ball, J.-M. K. Porter, and H. P. Siy, "If Your Version Control System Could Talk ...," in *ICSE Workshop on Process Modeling and Empirical Studies of Software Engineering*, 1997.
- [14] D. Cubranic and G. Murphy, "Hipikat: recommending pertinent software development artifacts," *25th International Conference on Software Engineering, 2003. Proceedings.*, vol. 6, pp. 408–418, 2004.
- [15] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proceedings of the 26th International Conference on Software Engineering, ICSE '04*, (Washington, DC, USA), pp. 563–572, IEEE Computer Society, 2004.
- [16] G. Canfora and L. Cerulo, "Impact analysis by mining software and change request repositories," *Proceedings - International Software Metrics Symposium*, vol. 2005, no. Metrics, pp. 261–269, 2005.
- [17] Xiaoxia Ren, B. Ryder, M. Stoerzer, and F. Tip, "Chianti: a change impact analysis tool for Java programs," *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pp. 664–665, 2005.
- [18] R. Chatley and L. Jones, "Diggit: Automated code review via software repository mining," *25th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2018 - Proceedings*, vol. 2018-March, pp. 567–571, 2018.
- [19] T. Mens and M. Goeminne, "Analysing the evolution of social aspects of open source software ecosystems," *CEUR Workshop Proceedings*, vol. 746, no. January 2011, pp. 1–14, 2011.
- [20] A. Murgia, P. Tourani, B. Adams, and M. Ortu, "Do developers feel emotions? an exploratory analysis of emotions in software artifacts," in *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, (New York, NY, USA), pp. 262–271, ACM, 2014.
- [21] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? an empirical case study of windows vista," in *Proceedings of the 31st International Conference on Software Engineering, ICSE '09*, (Washington, DC, USA), pp. 518–528, IEEE Computer Society, 2009.
- [22] S. K. Sowe, L. Angelis, I. Stamelos, and Y. Manolopoulos, "Using repository of repositories (rors) to study the growth of F/OSS projects: A meta-analysis research approach," in *Open Source Development, Adoption and Innovation, IFIP Working Group 2.13 on Open Source Software, June 11-14, 2007, Limerick, Ireland*, pp. 147–160, 2007.
- [23] Cerberus 2011-2019, "An open source, user friendly, automated testing tool." <https://www.cerberus-testing.org/index.php/en/>. Online; accessed 01 March 2019.
- [24] C. Calcagno and D. Distefano, "Infer: An automatic program verifier for memory safety of c programs," in *Proceedings of the Third International Conference on NASA Formal Methods, NFM'11*, (Berlin, Heidelberg), pp. 459–465, Springer-Verlag, 2011.

Software Engineering Risks from Technical Debt in the Representation of Product/ion Knowledge

Stefan Biffel¹, Lukas Kathrein^{1,3}

¹*Inst. for Information Systems Eng.
Faculty of Informatics
TU Wien, Vienna, Austria
[first].[last]@tuwien.ac.at*

Arndt Lüder²

²*Inst. of Ergonomics
Manufacturing Sys. and Automation
OvG U. Magdeburg, Germany
arndt.lueder@ovgu.de*

K. Meixner^{1,3}, M. Sabou^{1,3},

L. Waltersdorfer^{1,3} D. Winkler^{1,3}
³*Christian Doppler Lab SQI (sqi.at)
TU Wien, Vienna, Austria
[first].[last]@tuwien.ac.at*

Abstract—In the multi-disciplinary *production systems engineering* (PSE) process, software engineers depend on requirements and design rationales coming from product and production process planning, summarized as *product/ion* knowledge. Unfortunately, the engineering artifacts coming from product/ion planning often represent important product/ion knowledge incompletely and not well integrated, leading to risks regarding software engineering quality. In this paper, we report on a case study at a large industrial PSE organization, investigating *Technical Debt* (TD) effects, items, and causes in PSE process documentation and configuration management according to the VDI guideline 3695 Part 2. We focus on requirements for and issues in the representation of product/ion knowledge in the engineering data provided to software engineers. Based on data elicited from PSE domain experts, we model TD concepts based on the *Quality Function Deployment* method as foundation for TD analysis and risk management. The initial validation with domain experts revealed how software engineers could benefit from improved product/ion knowledge modeling as foundation for better understanding the rationale of engineering design decisions.

Keywords—Multi-Disciplinary Production Systems Engineering, Product/ion Knowledge, Product-Process-Resource (PPR) Model, Process Management, Technical Debt

I. INTRODUCTION

In *production systems engineering* (PSE) organizations, many different engineering disciplines work together, such as basic system and process planners, detailed automation engineers and production optimizers, for fulfilling customer requirements towards the *Industrie 4.0* vision [1] regarding, for example, production system throughput and quality. In a typical PSE process, the domain experts work in parallel in discipline-specific workgroups that exchange engineering artifacts for iterative improvement. For making informed design decisions, industrial automation and software engineers depend on the high quality of input artifacts that contain software requirements as well as results and rationale of system design decisions [4][5].

Unfortunately, the quality of *software engineering* (SE) results, such as software code governing the transportation system of a production plant, is subject to risks due to the missing or incorrect representation of product/ion knowledge, i.e., knowledge on characteristics of the product, produced by the plant or characteristics of the industrial production process and their relationships to characteristics of the production system [13].

An example for such a relation is a *fragile product* that imposes limitations on the maximal acceleration during the transport between production system components. If a software engineer sets the transport speed high to maximize the system throughput, the product quality may suffer, leading to the costly redesign of the overall system. Limited awareness of domain experts on the knowledge requirements of partner roles in the project may lead to insufficient descriptions of relevant engineering data and knowledge. Risks from decisions based on insufficient and often incomplete information or from unplanned effort due to unreliable communication between basic and detailed planners could be better managed with adequate knowledge representations of product/ion knowledge throughout the process. Kathrein *et al.* point in [13] out that *engineering organizations* (EOs), as defined in the VDI 3695 [28], tend to focus on discipline-specific outcomes rather than on the collaboration of domain experts. The domain experts suffer from low quality of collaboration artifacts, but do not, in general, have the knowledge or the budget to improve the collaboration process.

In this paper, we investigate *Technical Debt* (TD) in the representation of product/ion knowledge in engineering artifacts exchanged between PSE workgroups as foundation for analyzing and managing risks from TD effects, items, and causes in a PSE organization. An example for TD is a missing or incomplete engineering process description, which makes it hard to manage projects across several domains and work groups. In this paper we adapt the definition of TD by Li *et al.* [16] according to engineering artifacts and the collaboration process: *TD are violations in engineering artifacts compared to best practices of engineering process documentation and configuration for collaborative workgroups in the PSE domain*. Main goal is to identify TD throughout the engineering process, for better PSE process management, in particular, SE risks.

We report on results from a case study at a large industrial PSE organization on TD regarding process documentation according to the PSE domain VDI Guideline 3695 Part 2 [28] concerning the *procedure model for project activities and configuration management in an engineering organization*. We focus on eliciting requirements for and in the representation of product/ion knowledge supporting software engineers in their decision-making process. In the case study, we identified TD items, where one TD item is a unit bearing quality risk [16], on insufficient description of engineering process and information in the data exchange process and insufficient representation of product/ion knowledge. Based on collected data samples, we relate TD concepts to each other and investigate their relationships

DOI reference number: 10.18293/SEKE2019-037

based on the *Quality Function Deployment (QFD)* [19] method. The *QFD* method allows analyzing and prioritizing customer requirements together with solution options. We use the *QFD* method for analyzing TD repayment options [19], e.g., which TD items should be addressed to reduce system design cost. Better understanding TD relationships is the foundation for advanced analyses of TD risks and TD repayment options.

The remainder of this paper is structured as follows: Section II summarizes related work on PSE, on knowledge representation in PSE, and on concepts of TD in PSE knowledge. Section III introduces the research questions and approach. Section IV reports case study findings regarding TD effects, items, and causes, and relates these TD concepts in a preliminary *QFD* style model. Section V discusses results and limitations of the research. Section VI concludes and provides an outlook on future research.

II. RELATED WORK

This section summarizes related work on *production systems engineering (PSE)*, on knowledge representation in PSE, and on concepts of technical debt in PSE knowledge.

A. Production Systems Engineering (PSE)

The engineering of production systems is a multi-disciplinary task involving various disciplines, such as mechanical, electrical, and software engineering [3]. The disciplines conduct a network of engineering activities where engineering decisions are taken and engineering data are created by engineers. The engineers use appropriate input data and engineering tools optimized for the discipline. One role, the automation engineering designs and implements the hardware and software of the production system control, a main software engineering task in PSE [27].

Within PSE, the importance of digital models is increasing. New activities related to the development and use of life cycle crossing digital shadows of complete production systems and their components are envisioned to enable the *Industrie 4.0* vision [17]. These models shall contain all relevant data and knowledge on production systems aspects. This includes the description of the involved production system components, the production processes they execute, and the product resulting from the production process. Schleipen *et al.* [24] calls this *PPR knowledge* and Kathrein *et al.* [13] use the term *production knowledge*. In this paper, we build on the *PPR* concept to identify shortcomings regarding knowledge representation that introduce risks to SE activities.

B. Knowledge Representation in PSE

Engineering knowledge is a specific kind of knowledge, oriented towards the production of artifacts, and, as such, requires knowledge modeling and representation approaches that differ from other types of knowledge, such as taxonomical knowledge characteristic for the life sciences domain [25]. Ontologies are

information artefacts that have been used extensively to explicitly represent such engineering knowledge. This is for example investigated by Ekaputra *et al.* [7] highlighting different ontology-based data integration strategies, possible objectives an engineering organization has, as well as data source types used.

Sabou *et al.* [23] provide an overview of such ontologies and classify them in terms of the aspects of the PPR process that they cover. For example, OntoCAPE¹ [20] is an ontology for supporting *computer-aided process engineering (CAPE)* focusing on describing production process information. The *Semantic Sensor Network (SSN)*² ontology, developed at W3C³, is well suited to describe process states and their observability, as well as resource states. The *Automation Ontology (AO)* captures knowledge about industrial plants and their automation systems to support engineering simulation models [21]. AO concerns mechatronic concepts to support simulation model design and integration.

The explicit modeling of PSE knowledge is characterized by the need to address recurring modeling needs specific for this domain, including: Modeling *Part-whole relations*. Legat *et al.* [14] observe that containment hierarchies are a well-accepted and frequently occurring organizational paradigm from modeling part-whole relations in (mechatronic) engineering settings. Modeling *connections between components*. Legat *et al.* [14] observe that *interface-based composition* describes the capabilities expected from an interface to enable reasoning tasks about the correctness of a system's structure.

The modeling of recurring knowledge structures can be well addressed by the reuse of *Ontology Design Patterns (ODPs)* that model best practices applicable to typical conceptualization scenarios [10]. Indeed, ODPs exist to support the conceptual modeling of (variations) of the engineering-specific modeling scenarios mentioned above. For example, modeling *Part-whole* relations can be achieved by reusing the *PartOf* ODP⁴, which allows modeling part-whole relations in a transitive fashion. The *Componency* ODP⁵ is a specialization of the *PartOf* ODP for modeling *part-whole* relations distinguishing between direct and indirect (i.e., transitively-assessed) parts of an object. While there are several approaches for knowledge representation in PSE they are often not used and lead thus to TD, which is addressed in this paper.

C. Technical Debt in PSE Knowledge

Avgeriou *et al.* [2] compare *Technical Debt (TD)* to friction in mechanical devices, requiring increasingly more energy to achieve the same results as parts deteriorate. This is also true for *software engineering (SE)*, as short-term gains create friction over the lifetime of a software-intensive system that require extra effort and cost to address or to repay. To deal with TD, Avgeriou *et al.* [2] propose to analyze TD repayment options and to investigate TD from different viewpoints. TD in a system consists of TD items that are measurable in an SE artifact. Li *et al.* [16] identify ten different TD types, with effects ranging from

¹ OntoCAPE: <https://www.avt.rwth-aachen.de/AVT/index.php?id=730>

² SSN Ontology: www.w3.org/2005/Incubator/ssn/ssnx/ssn.owl

³ <https://www.w3.org/>

⁴ PartOf ODP: <http://ontologydesignpatterns.org/wiki/Submissions:PartOf>

⁵ Componency ODP: <http://ontologydesignpatterns.org/wiki/Submissions:Componency>

inconveniences to crippling whole software systems, making future maintenance costly [2]. Martini *et al.* [18] point out that large SE companies invest a quarter of the development time in managing TD to continue providing their SE functions.

Dong and Vogel-Heuser [6] draw a comparison, based on results from two case studies, between TD in PSE and in SE, as similar effects, such as short-term cost savings or lack of experience, occur in both domains. Causes of TD manifest in various dimensions, for example mechanical, electrical, or software engineering [6]. As process improvement and data exchange processes are success-critical processes in *engineering organizations* (EOs), they identify crucial TD in design and architecture, knowledge distribution and documentation [6].

Martini *et al.* [18] show how architectural TD accumulates during development in a project until reaching a crisis point that makes refactoring inevitable, increasing business value as the short-term sins are repaid adequately. Case studies by Biffel *et al.* [4][5] and Kathrein *et al.* [12][13] investigated engineering processes of EOs with a focus on the structure of collaborations between workgroups [13] and how data is exchanged [4][5]. These works represent building blocks for this paper, as they define a coherent context with basic concepts needed for TD investigations. The research highlights multiple use cases with different levels of TD, and points out missing *product/ion-aware* (PPR) knowledge as a limitation.

III. RESEARCH QUESTIONS AND APPROACH

This section introduces the research questions (RQs) following the *design science* method [29], and presents an illustrating use case to investigate TD in the representation of product/ion knowledge. Similarly, as in [26], we investigate TD as a form of software engineering risks, with effects and possible causes.

RQ1: What risks to software engineering results and activities are related to technical debt in the representation of product/ion knowledge in engineering artifacts exchanged between workgroups in production systems engineering? From the high-level RQ1, we derive the following sub-RQs.

RQ1a: What are effects of TD related to software engineering risks in PSE? We identify TD effects in the PSE process in interviews with domain experts. These TD effects can be defined as process management issues, i.e., deviations from the planned engineering process, and the process executed by individual domain experts. The identification of TD effects allows highlighting risks known to SE, but not to domain experts in PSE.

RQ1b: What are TD items regarding the VDI Guideline 3695 Part 2 in engineering artifacts exchanged between workgroups in PSE? The VDI Guideline 3695 Part 2 [28] provides valuable insights in describing engineering organizations (EOs) and potential improvement steps. The guideline provides a set of best practices that should be followed in an EO and allows analyses similar to code reviews in SE. Therefore, we define TD items in the PSE process by comparing selected target states in the VDI 3695 to the *as-is engineering process*.

RQ1c: What are causes regarding elicited TD items? As foundation for managing TD, we elicit in the case study candidate TD causes in the engineering organization. TD causes

strengthen the deviation between the as-is and VDI 3695 defined process and are important to address TD items and SE risks.

RQ2: How do TD concepts in the data exchange process relate to each other? After identifying TD concepts, we model their relationships as foundation for analyzing the impact of TD causes on TD items and effects, with a focus on SE concepts.

RQ2a: How do TD effects and TD items relate to each other? Main outcome of this RQ is a table based on the *QFD* method [19], developed with quality managers, who are responsible for defining an ideal PSE process across all involved disciplines and for possible improvement steps. The *QFD* method facilitates prioritizing relationships between TD effects and TD items that are relevant to reduce SE risks.

RQ2b: How do TD items and causes relate to each other? There are many and diverse TD cause candidates that can have different impacts on TD items. This RQ investigates most relevant TD causes to influence the TD items and effects. Main outcome is a table depicting relationships of TD causes and items based on the *QFD* method [19], created with quality managers.

To answer the RQs, we followed a case study design [22] by adhering to the following case study plan. [*Objective*] Exploring an existing engineering process [*Case*] in a large PSE organization. [*Theory*] Following the design science cycle according to Wieringa [29] in a holistic case study, [*Goal*] we identify common concepts at collaboration interfaces between PSE workgroups, and identify information bottlenecks regarding TD effects. [*Method*] Through seven semi-structured interviews (in a funnel approach) [22], [*Selection*] we elicit representative data from domain experts and investigate TD effects, items, and causes.

According to the design science cycle [29], this paper focuses on workshops and interviews regarding TD effects, causes, items, and the types and strengths of the relationships between the TD concepts. We discuss likely causes for the TD items found. Based on Matook and Indulska [19], we adapt the *QFD* method to focus in this paper on two dimensions of the *QFD House of Quality* (see Section V). In cooperation with quality managers, responsible for improving the PSE process, we design tables based on the *QFD* method [19] for investigating TD cause candidates. Finally, we present a conceptual evaluation, discussing presented repayment options to address SE-relevant TD in the multi-disciplinary engineering process.

Kathrein *et al.* [12][13] elicited the illustrating use case in Fig. 1 for *data exchange in the PSE process*. In this paper, the use case frames the election of TD concepts in the case study. In the beginning, the *system planner* (SP) receives product specifications from the customer (1) and aims at providing a competitive offer and at deriving specific knowledge on the production system for later use. This process is similar to software architecture design [14]. Output of this step (lilac arrow) are resource documents regarding the plant layout, calculations, and assembly sequences, delivered to the *process planner* (PP) (2).

Upon receiving the artifacts, the *PP* investigates these artifacts with a common schema that domain experts have developed over decades. For example, the first column always is the module identifier followed by the module name and a reference to an existing CAD drawing if possible. Main goal is to derive

The *production optimizer (PO)* receives all basic plans and tries to minimize the cycle times of the plant. However, this work requires different product/ion knowledge aspects and may cause many calls back to the *SP* and *PP* (5). The *PO* collaborates with the *automation engineer (AE)* (6), who is responsible for *PLC software engineering* tasks. From basic plans (4), the *AE* derives specific PLC software code. Goal of the *AE* is to trace design decisions as foundation for making informed design variations, such as the parameterization of the software and systems that execute production processes, e.g., the speed and acceleration of transport processes. In the next section we investigate this case study regarding TD effects, items and causes.

This section reports on findings from the case study regarding TD effects, items, and causes, and relates these TD concepts in a preliminary QFD model for the case study context.

TD effects. Regarding the use case, data exchange process, the following TD effects came up frequently in workshop sessions.

TD-E2 Data quality risks in engineering artifacts. Low quality of engineering artifacts may limit the production system capabilities and reduce reuse opportunities of system components.

Similar TD concepts in SE are missing documentation of interfaces, code, and implementation details.

Definition. The VDI Guideline 3695 Part 2 [28], configuration management, defines the target state A as “... *there are discipline-specific procedures for configuration identification, configuration monitoring, [...] Within a discipline, all employees follow common guidelines.*” However, in the case study, domain experts found artifacts not to be managed, but simply to evolve over time according to engineering personnel experience, without specific consideration for dependencies between engineering artifacts in different disciplines, which poses risks for SE activities that depend on consistent and complete inputs (see Fig. 1).

Measurement. There is no formal description of engineering artifacts and data, including dependencies between engineering disciplines, such as product, process, and system design. There is no configuration history for backtracking design decisions.

Relationships to effects. See Table I, in Section IV.C. TD symptoms include high effort and risk for propagating changes to systems design across disciplines, in particular for SE, when receiving inputs from several engineering disciplines.

Relationships to causes. See Table II, in Section IV.C.

Product/ion (PPR) knowledge representation insufficient (TD3PPR) (in exchanged engineering data). *Motivation.* Domain experts in production process design have product/ion (PPR) knowledge that would be, in many cases, important to engineers in later stages of PSE and optimization, in particular, for SE activities. However, the process designer tends to provide her engineering partners with hard-coded production system parameters rather than PPR knowledge as there is no dedicated tool or modeling language to allow the effective and efficient representation of PPR knowledge. Short-term benefit for the process designer is saving effort for modeling the PPR knowledge. Similar TD concept in SE would be missing information on non-functional requirements for a software system.

Definition. The VDI Guideline 3695 Part 2 [28], configuration management, defines the target state D as “*system-assisted cross-discipline*” configuration management to enable “*consistency check [...] at an early stage*”. However, without sufficient PPR knowledge representation, consistency checks between production process design and production system design are difficult, error-prone, and take considerable expert effort.

Measurement. The engineering data model misses representations for expressing PPR knowledge and rationale to trace design decisions, such as production system temperature settings to the welding temperature and force of a metal joining process.

Relationships to effects. See Table I, in Section IV.C. TD symptoms include in SE activities considerable costs of errors from changes and effort for preventing defects after changes.

Relationships to causes. See Table II, in Section IV.C. In the case study context, main cause candidates include workgroup-specific optimization of the engineering organization and insufficient means to express PPR knowledge.

C. Case Study results on TD cause candidates (RQ1c)

Cause candidates linked to context in the engineering organization, often for economic and historic reasons in the EO.

A1. Workgroup-related profit centers lead to the local optimization of workgroups with limited concerns for the optimization of projects across workgroups, often at the expense of SE.

A2. Engineering habit trained by discipline-specific education leads to engineers focusing on good results in their workgroup. Engineers are, in general, not aware about work tasks, dependencies, and problems in other workgroups, unless a partner asks them for an improvement.

A3. Unclear responsibilities of domain experts in data exchange process lead to ad-hoc procedures and data definitions.

A4. Limited collaboration effort across work groups without a dedicated role for coordinating the work across workgroups.

Cause candidates from engineering process description

B1. Engineering process modelled as an artifact-based workflow, not as a data-related workflow makes it hard to describe dependencies between SE and other disciplines, such as consistency rules that relate to the data model, not to artifacts.

B2. Engineering process defined, but not useful. There is a workflow definition for a process. However, the definition may be abstract and lack important description of content dependencies, such as relationships between the product and resource design, tainting the usefulness of the definition.

B3. Engineering process defined, but not operational. There is a process description. However, missing technical foundations, such as adequate process description concepts or tool support, make it hard or impossible to conduct the process.

B4. Engineering process defined, but not known to stakeholders. There is a process description somewhere in a manual. However, the relevant actors in the project are not aware of the process description for their daily work.

Cause candidates linked to information description

C1. Description of complex dependencies required due to a large number of disciplines (often 15 or more) in a PSE project. Complex descriptions of processes and artifacts and their dependencies in an *engineering organization* (EO) lead to a very complex network (consider Fig. 1, scaled up).

C2. Industry-dependent information description. The description of information depends on the industry and has to be adapted accordingly. There is no general standard that could be applied directly. There is no general EO model as the industry domains require a variety of EO structures and behavior

C3. Tool-driven process without product/ion (PPR) information description. Often, the process is defined based on a specific tool chain. Therefore, the functional and data export capabilities of the tool determine the exchanged information. The process is not aware of PPR as the discipline-specific tools only know the PPR knowledge that is relevant within the discipline.

D. TD effect and item relationships (RQ2a)

Following an adaptation of the *QFD* method according to Matook and Indulska [19], we create a *House of Quality (HoQ)*. Our *HoQ* provides insights into the relationships between TD effects and items horizontally (representing customer requirements in the original *HoQ*) and TD items and causes in the vertical axis (representing engineering requirements). For these two *QFD* dimensions, we design two tables expressing likely correlations and relationships. We elicited and aggregated likely relationships from a workshop with domain experts in the exploratory case study context [12][13]. As relationship types differ, we indicate the following types and strengths. *DS* indicates a *direct* and *strong* relationship (the stronger the item, the stronger the effect). *DW* indicates a *direct weak* relationship (a stronger item correlates moderately to a stronger effect), and *IW* expresses an *indirect weak* relationship (stronger cause leads to a lower TD item). *No* indicates that the TD item is not related to an effect, such as (*TD1Proc*) -> (*II. Data Quality Risk*).

RQ2a. Table I presents the relationships between TD effects (see Section IV.A) and TD items (see Section IV.B), similar to the *HoQ* analysis [19] matrix, TD effects horizontally and TD items in the vertical axis. The relationships are of the form TD item relates to TD effect, (*TD item*) -> (*TD effect*), expressing how a TD item relates to an effect.

Table I. Relationships between TD effects and TD items.

TD Effect/ TD Item	TD-E1 High Effort	TD-E2 Data Quality Risk	TD-E3 Economic Failure
TD1Proc	DS	No	DS
TD2Inf	DS	DS	DW
TD3PPR	DS	DW	DW

Legend: Relationships: DS: direct strong; DW: direct weak.

In Table I, all three TD items, relate strongly to the TD effect *TD-E1 High Effort* for SE. This is due to unclear descriptions of the process and information as well as missing product/ion knowledge, which all lead to high effort for tracing design decisions. An insufficient information description relates strongly to high risks in data quality, as artifacts are not built on common concepts or data models and thus lack any formal description. Missing product/ion knowledge is also related to the second TD effect (*TD-E2*), however not so strongly. All three TD items have a relation to the *TD-E3 Economic Failure*, as missing information in the engineering process leads to high rework and communication overheads.

E. TD item and cause relationships (RQ2b)

Table II represents the relationship between TD items (see Section IV.B) and TD cause candidates (see Section IV.C), (TD cause) -> (TD item). Cause candidates coming from the context of the EO (A1 – A4), and from the engineering process description (B1 – B4) have a strong direct relationship to the engineering process description (*TD1Proc*). For example, unclear relationships and descriptions which are not useful, make it very hard to describe the engineering process sufficiently to facilitate collaboration and coordination across multiple workgroups.

All three cause groups A, B, and C relate to the insufficient description of the engineering data exchange model (*TD2Inf*). Note the *inverse relationships* of a stronger focus on engineering

habits (intra process improvements) and descriptions of the engineering process as artifacts. This does not directly impact the TD item.

Table II. Relationships between TD items and TD causes.

TD Item -> TD Cause (see Sect. IV.C)	TD1 Proc	TD2 Inf	TD3 PPR
A1.Profit Center	DW	DS	DS
A2.Engineering Habits	DW	IW	DS
A3.Unclear responsibility	DS	DW	DS
A4.Limited collaboration	DW	DS	DS
B1.Eng. Proc. descr. as artifact	No	IW	DS
B2. Eng. Proc. descr. not useful	DS	DW	No
B3. Eng. Proc. not operational	DS	DW	No
B4. Eng. Proc. unknown	DS	No	No
C1.Inform. description. complex	DS	DW	No
C2. Inf. desc. Industry. depend.	No	DS	DW
C3.Tools w/o PPR	No	DW	DS

Legend: DS: direct strong; DW: direct weak; IW: indirect weak.

Insufficient descriptions of the engineering process make it impossible to successfully represent product/ion-aware knowledge (*causes Ax*) -> (*TD3PPR*). Causes regarding the information and data exchange description do not impact product/ion knowledge representations, as a major precondition for knowledge representation is the clarification of (a) the responsibility for each part of product/ion knowledge and (b) a suitable represented approach throughout an engineering process.

F. Preliminary validation in the exploratory case study

Throughout the domain expert interviews, we collected representative data samples from engineering artifacts. We derived tables I and II from analyzing these artifacts. As the tables present vital pieces of information regarding possible correlations, we initially elicited the relationships from domain experts. We discussed the relationship candidates in detail with quality managers, who are responsible for improving the engineering process and are knowledgeable in the overall process and work group habits, including SE. We resolved divergences between the views of the domain experts and the quality managers in a common discussion. Overall, the domain experts and quality managers found the preliminary TD concepts and analysis method useful and usable for identifying and addressing high-priority TD effects, items, and causes regarding SE activities.

V. DISCUSSION

This paper investigates risks for *software engineering* (SE) in activities related to the engineering process in a PSE organization (*RQ1*) and possible relationship between certain risks (*RQ2*). In this context, risks are TD effects for SE that occur in a PSE context with measurable probability and costs. To deal with these risks, da Luz *et al.* [26] presented a management tool for analyzing causes and effects. Similar to our work, Luz *et al.* [26] propose an approach to identify risks through selection, description and analyzation. However, the presented approach generalizes risks in a late phase, whereas we focus on organization specific TD effects and investigate these. In the exploratory case study context, SE activities depend on the requirements and design rationale from early engineering phases and often have to deal with locally optimizing workgroups, low awareness on collaboration processes, and missing understanding of requirements between work groups.

For software engineers, the *high effort* comes from risks regarding rework efforts due to frequent and late changes coming from earlier phases. As software engineers highly depend on weakly documented design decisions from early phases, a repayment option for TD is better knowledge representation of product/ion knowledge throughout the engineering process. The *low data exchange quality* impacts software engineers, who are made responsible for low quality system output, even if they write high quality code, but based on weakly communicated early design decisions. TD repayment for reducing the SE risk should focus on improving the documentation and communication of design decisions that are directly related to high-quality SE results. Finally, issues regarding unplanned efforts for reworks in the software design due to low system quality decisions may exceed the budget available to SE, leading to local *economic failure*.

The VDI Guideline 3695 Part 2 [28] was used to investigate TD items (*RQ1b*). This guideline can be seen similar to best practices for SE code development, and our analysis is equivalent to a code review. An *unclear engineering process description* makes it hard for software engineers to reliably configure production systems, as input from several disciplines may be contradicting. The missing collaboration in PSE forces software engineers to take the risky decision on which inputs to consider or ignore. Further, the *information description is not sufficient*. This makes it unclear for software engineers where to look for reliable information, as data syntax and semantics may change frequently, making it hard to validate input data and to automate the data exchange process. Software engineers thus often work based on risky assumptions. Finally, *missing product/ion aware knowledge* makes it hard in SE to take informed decisions for adapting the software system design if the preferred system design option is not feasible.

In *RQ1c*, we investigated possible causes regarding elicited TD items. Causes linked to the *context of production systems engineering* cannot be directly influenced by SE actors, but require the insight of PSE managers. We discussed the preliminary results at the case study EO with quality managers, who found the analysis useful for considering and prioritizing improvement options. Cause candidates linked to the *engineering process description* clearly motivate the need for better means of PPR knowledge representation as a foundation for process descriptions, considering conceptual, language, and usability aspects. This is similar to the need for proper software architecture descriptions identified by Guessi *et al.* [11]. The last group we identified are cause candidates linked to the *information description*. For example, it is challenging to combine methods for data integration [7] with domain-specific standards, such as *AutomationML* [7] or ontologies [23].

A *repayment option* to address TD items related to weak collaboration of workgroups is a new role, the *data curator*, similarly as presented in [9]. This role would be responsible for consolidating a common data exchange model and describe the engineering process adequately. This new role should needs to understand the requirements and limitations of the involved workgroups, in particular SE activities. As TD effects, items and causes are related to each other, we used the *Quality Function Deployment* [19] method to investigate relationships between TD effects, items and causes (*RQ2*).

In *RQ2a* we investigated how TD effects and TD items relate to each other. The *TD-E1 High Effort* is strongly and directly connected to all three TD items. This is obvious as reworks are often needed to compensate missing descriptions or information bottlenecks where especially software engineers are affected. Interesting is, that the *TD-E2 Data Quality Risk* is only weakly connected to PPR knowledge representation, even though this is an important information exchange concept. For SE this means that design decisions from early phases do not impact the code development so much as the overall information description, this could be for example the selection of an easily changeable component in the user interface.

RQ2b investigated how TD items and causes are related. Nearly all causes regarding the information description strongly impact the process description. This clearly motivates the need for better knowledge representation approaches, as the current engineering process is either not described, or the description is not useful or unknown to domain experts. As there are currently no tools that support the expression of PPR knowledge, software engineers could address this open issue to improve product/ion-aware knowledge representation and further allow a backflow of SE knowledge into early engineering phases as foundation for designing better reusable system parts.

Limitations. The research of this paper followed a case study in an engineering organization. However, the case study focused on only one company, which may not be representative for all EOs in general. While the domain experts in the study were very knowledgeable, their number was limited due to resource limitations of the available experts besides their daily business obligations. We found that the engineering process description may highly depend on the context, domain, and organization, thus future case studies should consider these variation points. While the domain experts found the preliminary list of TD effects, items, and causes, and their relationships useful for reflecting on TD repayment options in the case study context, these results require validation and discussion on comparable context for strengthening the external validity of the results.

VI. CONCLUSION AND FUTURE WORK

In engineering organizations, software engineers join the PSE process in a late phase and are concerned with detailing SE aspects of the software-intensive system. However, software engineers often only receive poorly described design decisions in form of engineering artifacts making it hard for them to derive adequate new (software) engineering knowledge or tracing the earlier design decisions. These shortcomings lead to risks regarding the SE quality and impact the project effort negatively, endangering project success. In this paper, we reported on a case study at a large industrial engineering organization with the focus of investigating technical debt (TD) effects, items, and causes as risks for software engineers. Results highlight that TD can slow down engineering organizations, making it hard to manage processes where multiple domains are involved. Main insight for addressing the found challenges is the introduction of a new role, the *data curator*, to facilitate the collaboration across workgroups. The results highlight requirements for the representation of product/ion knowledge in the engineering data provided to software engineers. Engineering data is heterogeneous and there are no guidelines for basic planners, leading to a large

number of individual and local data models, and making the data exchange hard to manage, often resulting in extra effort to maintain high software quality.

The relations between TD effects, items, and causes highlighted the need for better representations for product/ion knowledge as inadequate context and artifact descriptions lead to high efforts, in particular, for software engineers, and might result in economic project failure. The research findings provide domain experts, such as project managers or software engineers with insights into the engineering process. The presented model serves as a foundation for better understanding the rationale of engineering design decisions. This leads to better SE code due to (a) better understanding of design decisions, (b) more explicit representation of system limits that relate to product characteristics, and (c) better knowledge representation for tool support.

Future work. The results of the exploratory case study should be validated with empirical data from comparable engineering companies. We focused in this paper on product/ion-aware exchange of engineering artifacts. Future research should investigate the impact of knowledge representation options on selected SE. Finally, the more comprehensive representation of integrated PSE knowledge requires improved information security. The comprehensive and well-integrated knowledge is a prime target for attackers regarding corporate espionage and regarding the intentional change of artifacts for reducing the quality of the production system or the production process. Thus, future work should investigate security auditing aspects that consider the issues and repayment options identified in this paper.

ACKNOWLEDGMENT

The financial support by the Christian Doppler Research Association, the Austrian Federal Ministry for Digital & Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

REFERENCES

- [1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
- [2] P. Avgeriou, P. Kruchten, R. L. Nord, I. Ozkaya, and C. Seaman, "Reducing friction in software development," *IEEE Software*, vol. 33, no. 1, pp. 66–73, 2016.
- [3] S. Biffl, A. Lüder, and D. Gerhard, *Multi-Disciplinary Engineering for Cyber-Physical Production Systems: Data Models and Software Solutions for Handling Complex Engineering Projects*. Springer, 2017.
- [4] S. Biffl, A. Lueder, F. Rinker, L. Waltersdorfer, and D. Winkler, "Introducing engineering data logistics for production systems engineering," Technical Report CDL-SQI-2018-10, TU Wien; <http://qse.ifs.tuwien.ac.at/wp-content/uploads/CDL-SQI-2018-10.pdf>.
- [5] S. Biffl, A. Lueder, F. Rinker, L. Waltersdorfer, and D. Winkler, "Efficient Engineering Data Exchange in Multi-Disciplinary Systems Engineering," in *Proc. Int. Conf. on Advanced Information Systems Engineering (Caise)*. IEEE, 2019, in press.
- [6] Q. H. Dong and B. Vogel-Heuser, "Cross-disciplinary and cross-life-cycle-phase technical debt in automated production systems: two industrial case studies and a survey," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1192–1199, 2018.
- [7] R. Drath, A. Luder, J. Peschke, and L. Hundt, "Automationml the glue for seamless automation engineering," in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*. IEEE, pp. 616–623, 2008.
- [8] F. J. Ekaputra, M. Sabou, E. Serral, E. Kiesling, and S. Biffl, "Ontology-based data integration in multi-disciplinary engineering environments: A review," *Open Journal of Information Systems (OJIS)*, vol. 4, no. 1, pp. 1–26, 2017.
- [9] A. Fay, U. Löwen, A. Schertl, S. Runde, M. Schleipen, and F. El Sakka, "Zusätzliche Wertschöpfung mit digitalem Modell," *atp magazin*, vol. 60, no. 06-07, pp. 58–69, 2018.
- [10] A. Gangemi and V. Presutti, "Ontology design patterns," in *Handbook on ontologies*. Springer, pp. 221–243, 2009.
- [11] M. Guessi, F. Oquendo, and E. Y. Nakagawa, "An approach for capturing and documenting architectural decisions of reference architectures," in *Proc. SEKE*, pp. 162–167, 2014.
- [12] L. Kathrein, A. Lueder, K. Meixner, D. Winkler, and S. Biffl, "Process analysis for communicating systems engineering workgroups," Technical Report CDL-SQI-2018-11, TU Wien; <http://qse.ifs.tuwien.ac.at/wp-content/uploads/CDL-SQI-2018-11.pdf>.
- [13] L. Kathrein, A. Lüder, K. Meixner, D. Winkler, and S. Biffl, "Product/ion-Aware Analysis of Multi-Disciplinary Systems Engineering Processes", presented at 21st Int.l Conf. on Enterprise Information Systems, Heraklion, Greece, May 2019 (in press).
- [14] C. Legat, C. Seitz, S. Lamparter, and S. Feldmann, "Semantics to the shop floor: towards ontology modularization and reuse in the automation domain," *IFAC Proce. Volumes*, vol. 47, no. 3, pp. 3444–3449, 2014.
- [15] X. F. Liu, N. Chanda, and E. C. Barnes, "Software architecture rationale capture through intelligent argumentation," in *SEKE*, pp. 156–161, 2014.
- [16] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, 2015.
- [17] U. Loewen, A. Schertl, S. Runde, M. Schleipen, F. El Sakka, and A. Fay, "Additional value with a digital plant model-new roles over a plant's lifecycle," *ATP EDITION*, no. 6-7, pp. 58–68, 2018.
- [18] A. Martini, T. Besker, and J. Bosch, "Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations," *Science of Computer Prog.*, vol. 163, pp. 42–61, 2018.
- [19] S. Matook and M. Indulska, "Improving the quality of process reference models: A quality function deployment-based approach," *Decision Support Systems*, vol. 47, no. 1, pp. 60–71, 2009.
- [20] J. Morbach, A. Wiesner, and W. Marquardt, "Ontocapea (re) usable ontology for computer-aided process engineering," *Computers & Chemical Engineering*, vol. 33, no. 10, pp. 1546–1556, 2009.
- [21] P. Novák, E. Serral, R. Mordinyi, and R. Sindelár, "Integrating heterogeneous engineering knowledge and tools for efficient industrial simulationmodel support," *Advanced Engineering Informatics*, vol. 29, no. 3, pp. 575–590, 2015.
- [22] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, p. 131, 2009.
- [23] M. Sabou, O. Kovalenko, and P. Novák, "Semantic modeling and acquisition of engineering knowledge," in *Semantic Web Technologies for Intelligent Engineering Applications*. Springer, pp. 105–136, 2016.
- [24] M. Schleipen, A. Lüder, O. Sauer, H. Flatt, and J. Jasperneite, "Requirements and concept for plug-and-work," *at-Automatisierungstechnik*, vol. 63, no. 10, pp. 801–820, 2015.
- [25] M.-a. Sicilia, E. García-Barriocanal, S. Sánchez-Alonso, and D. Rodríguez-García, "Ontologies of engineering knowledge: Generalstructure and the case of software engineering," *The Knowledge Engineering Review*, vol. 24, no. 3, pp. 309–326, 2009.
- [26] D. da Luz Siqueira, L. M. Fontoura, R. H. Bordini, and L. A. L. Silva, "A knowledge engineering process for the development of argumentation schemes for risk management in software projects," in *Proc. SEKE*, pp. 36–41, 2017.
- [27] A. Strahilov and H. Hämmerle, "Engineering workflow and software tool chains of automated production systems," in *Multi-Disciplinary Eng. for Cyber-Physical Production Systems*. Springer, pp. 207–234, 2017.
- [28] VDI 3695: Engineering of industrial Plants, Evaluation and Optimization, Beuth Verlag Std., 2009.
- [29] R. J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.

A Enhanced Feature Model for Software Product Line and Core Feature Extraction

1st GuanZhong Yang

Department of Software Engineering
Hunan University
Changsha, China
gzyang@hnu.edu.cn

2nd HaoMing Chang

Department of Software Engineering
Hunan University
Changsha, China
haomingchang@hnu.edu.cn

3rd ZeYa Mou

Department of Software Engineering
Hunan University
Changsha, China
mouzey@hnu.edu.cn

Abstract—Feature model is an important model for capturing domain requirements and managing commonality and variability. However, the traditional feature modeling method is insufficient in expressing the variability requirements, which is easy to cause ambiguity and fails to describe the features in detail. In view of these shortcomings, this paper gives a unified definition of features, feature attributes and feature relationships, proposes configuration vector and variant constraint to enrich variability expression capabilities, and establishes an enhanced feature meta-model for software product lines. This feature model can help domain analysts effectively organize the variable domain requirements. In addition, the commonality of the domain expressed by the feature model is the basis for the development of highly reusable core assets. We propose that heuristic strategies automatically identify and extract core feature to obtain domain commonality. Finally, we take the chronic obstructive pulmonary disease home care product line as an example to demonstrate our proposed method.

Index Terms—feature modeling; software product line; core feature; commonality analyze

I. INTRODUCTION AND MOTIVATION

Software product line engineering is a paradigm for large-scale development of software applications [1]. The general idea is to plan and design reusable components for reuse before developing the system, and to manage the common and variable parts of the components [2]. As part of Feature-Oriented Domain Analysis(FODA) [3], feature model(FM) has proven to be an effective way to describe product line commonality and variability.

The Chronic Obstructive Pulmonary Diseases(COPD) Home Care System is a product line we are developing. The purpose of this application is to manage and monitor the stable COPD patients in home or community. It is designed to provide patients with environment monitoring, signs monitoring, doctor consultation, health education, acute exacerbation warnings and behavioral interventions. We have used FM to perform requirement analysis and domain analysis. Fig.1 uses the basic feature model(BFM) to demonstrate part of the features of the *signs monitoring* in the system.

Traditional feature modeling methods do not separate the perspectives of different stakeholders. The product line of

COPD home care involves services provided to users, domain knowledge related to COPD, various data monitoring devices and complex implementation technologies. If stakeholders share the same feature model view, they will not be able to effectively focus the perspective. Though extended-feature model [4]–[8] enrich the expression of variability from different perspectives, there is no agreement on the definition of modeling elements, especially two kinds of configuration information: mandatory and optional. This can lead to inadequate expression of variability and even semantic conflicts. Therefore, the traditional feature modeling methods are difficult to effectively satisfy the domain analysis requirements of our product line.

In addition, the COPD home care product line is a customizable application for individualized patients. The patient's use scenario and the patient's condition will bring more variability to the system. In the early stage of development, accurately extraction of core requirements in established domain model is required so that we can distinguish between common and variable requirements and build the basic platform based on core requirements in the subsequent development to speed up the time to market.

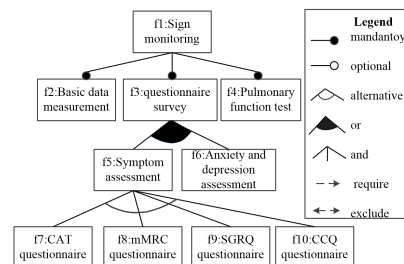


Fig. 1. Example of basic feature model.

In response to the above questions, this paper made the following contributions:

- Introduce an enhanced feature model meta-model. We classify features to form different levels of abstraction, propose configuration attribute vectors to resolve semantic conflicts and enrich the expression of model variability.

Foundation Items: The Science and Technology Department Project of Hunan China(2018TP2033)

DOI reference number: 10.18293/SEKE2019-110

- Based on our meta-model, a core feature extraction heuristic strategy is proposed. The purpose is to automatically analyze the commonality of product families on a consistent feature model.

The rest of this paper is organized as follows. Sect.II describes the related work. Sect.III introduces the meta-model modeling elements. Sect.IV describes the core feature extraction method. We use a practical case to verify our ideas in Sect.V. And we conclude our work in Sect.VI.

II. RELATED WORK

[7] introduced Forfamel, a rigorous conceptual model foundation that combines multiple feature modeling methods. A cardinality-based feature model(CFM) formally defined in [4]. CFM adds the concept of feature cloning, using feature-cardinality to represent the number of cloned features. Group-cardinality in CFM is used to express configuration constraint of variants in variation points. [3] considered that different features can be refined into the same feature, and proposed the concept of pseudo-feature. [6] classified feature into capability feature, operational environment feature, domain technical feature, and implementation technical feature. The FM was divided into four levels corresponding to feature classification which mapped to implementation domain.

Some works have been devoted to extracting core features. The concept of core feature proposed in [8]. [9] transformed feature model into binary constraint logic, automatically analyzed all valid product through the CSP solver and obtained the core features. [10] proposed an algorithm traverses BDD by backtracking method to determine the core features and dead features. Although the algorithm was very efficient, converting the feature model to BDD was time consuming and reached an exponential level.

III. THE ELEMENTS AND RELATIONS OF META-MODEL

Fig. 2 shows our meta-model. We will introduce the terms and concepts necessary to understand the meta-model.

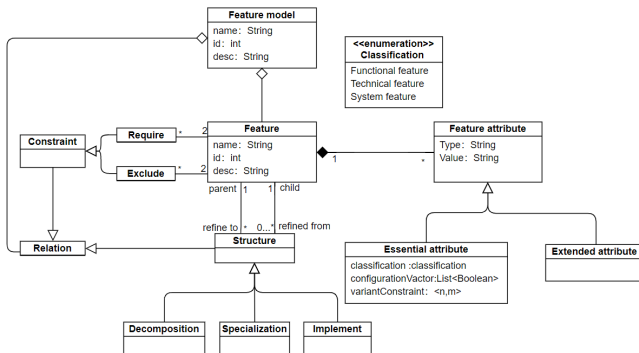


Fig. 2. Enhanced feature model meta-model.

A. Feature and Feature Attribute

Definition 1(Feature): A Feature f consist of a set of logically relevant software requirements that provide each stakeholder with the ability to satisfy specific business demand. A feature is defined as a 4-tuple: $f=(N,D,A,R)$, where: $N(\neq \phi)$ is the feature name to uniquely identify a feature. D is an explanation for the feature. A is a set of feature attributes, which describes the context information of the feature in a FM. R represents the relationship between the features.

We extend and represent the existing modeling elements with feature attributes. Feature attributes are divided into two categories:essential attribute and extended attribute. They are defined by a 2-tuple: $A=(T,V)$, where T is a set of attribute types for the feature, V is the value of the corresponding attribute type. Attribute of a feature can be expressed as: $f_A.type = value$. There are four types of feature attribute. $T:=\{Classification, Configuration, VariantConstraint, Extension\}$

Classification attribute. We classify feature into functional feature(FF), technical feature(TF) and system feature(SF).

- *Functional feature(FF)* is used to describe the services provided by the system, which can be clearly expressed and visible to users and is a specific behavior of the system.
- *Technical feature(TF)* is used to describe the technologies that implement functional features or non-functional indicators. Such features are not visible to the user, but developers are more concerned with these.
- *System feature(SF)* is used to describe the hardware resources, implementation environment of the system to facilitate communication between hardware and software engineers.

This categorization allows developers to refine different abstract level as they perform domain analysis and helps to separate the perspectives of different stakeholders. The expression $f_A.classification$ can be used to represent the classification attribute of feature f .

Configuration attribute. We attach optional and mandatory configuration information to the feature through feature attributes. Some domain feature modeling methods differ in their understanding of mandatory and optional feature. [5], [11], [12] gave an absolute definition of the mandatory feature, who considered that the mandatory features were the features appearing in all products in a product line, not related to its parent. However, [6], [7] interpreted it relatively, and considered that the mandatory feature must be selected when its parent feature was selected in the configuration process. We recognize and adopt the relative interpretation of the mandatory and optional, and introduces *core feature* to represent the absolute interpretation of mandatory feature. There are two drawbacks in the absolute definition of configuration attribute.

- It loses feature parent-child relationship contextual information.
- In the case that a feature has multi-parent features, the absolute definition is impossible to express the semantics

that child-feature's configuration attributes is different relative to different parent.

As shown in Fig.3(The specific features are represented by symbols.), feature $f1$ and $f2$ can be implemented by $f4$. The $f4$ is optional for $f1$ and mandatory for $f2$. When a feature have multi-parents, absolute interpretation of mandatory feature will cause ambiguous semantic. According to the definition of relativity in this paper, the configuration attribute of a feature may be different for different parent features, it is necessary to record these additional information which may change.

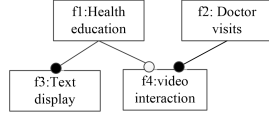


Fig. 3. Semantic conflict under absolute definition of configuration information.

Definition 2(Configuration vector): If the number of parent of a feature f is m and its parents are $(p1, p2, \dots, pm)$ from left to right in the feature diagram. Then C is a 1-dimensional vector of m components. $C := (bool_{p1}, bool_{p2}, \dots, bool_{pi}, \dots, bool_{pm})$

The component of vector C is boolean type, and $bool_{pi} = 1$ denotes that f is a mandatory feature relative to the i -th parent feature, otherwise it is an optional feature. We use $f_A.config = C$ to represent configuration attribute.

Variant constraint. This attribute represents the configuration constraint for all optional child-feature under a parent during configuration process. Similar to the group-cardinality in CFM. Fig.4 shows the feature *symptoms assessment*. In global initiative of COPD[13], we use $f2$ to evaluate the symptoms, and a comprehensive assessment is recommended using feature $f2$, $f3$ or $f4$ as supplement. Therefore, when selecting $f2$, at least one must be selected in $f3$, $f4$ and $f5$.

Definition 3(Variant constraint): $\langle n, m \rangle$ is an interval. The n is the lower bound and m is the upper bound. Let $k \geq 0$ be the number of optional child-features that satisfy $0 \leq n \leq m \leq k$. Variant constraint attributes of feature f indicate that if the feature is selected in configuration process, n to m optional features should be selected besides mandatory child-features. We use $f_A.variant = \langle n, m \rangle$ to represent configuration attribute.

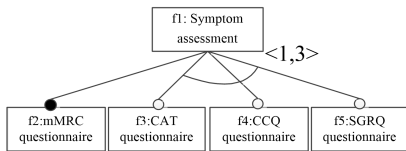


Fig. 4. Example of variant constraint.

Extension attribute. This attribute is a feature extended attribute whose type is defined by the domain analyst. Such as adding the binding time and binding status of the feature.

B. Feature Relationships

Features form feature model with certain relationships. R is a set of feature relations and $R = R_{struct} \cup R_{constraint}$. $R_{struct} \subseteq f \times f$ represents the parent-child relationship of the feature. If feature $f1$ and $f2$ satisfy $(f1, f2) \subseteq R_{struct}$, then $f2$ is the parent of $f1$, and $f1$ is the child of $f2$. We extend three types of relations over structure relation, $R_{struct} = R_{decomposition} \cup R_{specialization} \cup R_{implement}$.

$R_{decomposition}$ describes the whole-part relationship between parent and child features. In Fig.1, $f1$ is decomposed into $f2$, $f3$, and $f4$.

$R_{specialization}$ represents that a feature can be specialized from different dimensions or aspects into more specific child-features. Child-features have the characteristics of parent, similar to inheritance in object-oriented, and they have special characteristics on the basis of inheritance. In fig.1, $f5$ is specialized into $f7$, $f8$, $f9$, $f10$.

$R_{implement}$ represents the technical implement or hardware support of the low-level features to the high-level features.

$R_{constraint} \subseteq f \times f$ represents the constraint between the same classification attribute features and $R_{constraint} = R_{req} \cup R_{excl}$. If $(f1, f2) \subseteq R_{req}$ indicates that $f1$ requires $f2$. If $f1$ is selected during configuration, then $f2$ also needs to be selected. This relationship is transitive. $(f1, f2) \subseteq R_{excl}$ indicates that $f1, f2$ cannot be bound at the same time, which is symmetry.

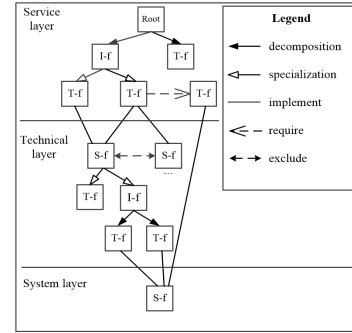


Fig. 5. Example of feature model structure.

Three classifications of features form three abstract levels of feature models. Fig.5 shows the basic structure of our FM diagram, where *root* is root of FM, *I-f* is the intermediate feature, *T-f* is the terminal feature, and *S-f* is the starting feature of the technical layer and the system layer. The above-mentioned structure relationships provide domain analysts with a way to construct a feature model. From root, the features are continuously refined through $R_{decomposition}$ and $R_{specialization}$. Then consider which technical features or system features are required to implement the service feature *T-f*. At this point, the technical feature or system feature is the *S-f* of the hierarchy, and then it can be considered whether these features can continue to be refined. By continuously iterating through this process, a FM diagram can be obtained. The relationship between $R_{decomposition}$ and $R_{specialization}$

makes the feature form the parent-child relationship of tree structure in the same layer, $R_{implement}$ connects T - f and S - f , thus connecting different abstract levels of FM, which becomes directed acyclic graph(DAG).

IV. ANALYSIS OF COMMONALITY

A. Core Feature Extraction Strategy

Definition 1(Core feature): Core feature is the feature that appear in all products which reflects the commonality of FM. If there are n valid products in the product line. Then core feature f_c are satisfied: $f_c \in \bigcap_{i=1}^n p_i$. Following rules are proposed to identify core feature:

Rule 1 The root feature in FM must be the core feature. Because the root appear in all products.

Rule 2 If there is a path starting from a core feature, and all the features on this path are mandatory for its parent, then these features are core feature. We call the core feature on this path directly accessible core features(DACF).

Rule 3 If there is a path starting from a core feature, and all the features on this path are linked by R_{req} , then these features are core feature. We call it the required transfer core feature(RTCF).

Rule 4 Indirectly accessible core features(IACF) that are easily overlooked.

Only extracting DACF and DTCF will result in incomplete core feature set. Fig.6 shows a special example (This figure is intended to make the example more intuitive, not the representation of our method.). Assume $f_1, f_{11}, f_{12}, f_{13}, f_{14}$ as FF , f_2 as TF . Feature f_{11}, f_{12} and f_{13} are all implemented by f_2 . Feature f_1 is the core feature, f_{11}, f_{12} and f_{14} are optional, f_{13} is mandatory, and the $f_{2A}.conf = (1, 1, 0)$. According to rule 2, only f_{13} is a core feature. However, because of $f_{1A}.variant = \langle 2, 2 \rangle$, one of the feature combination $\{f_{11}, f_{14}\}, \{f_{12}, f_{14}\}$ or $\{f_{11}, f_{12}\}$ must be selected. Therefore, f_{11} and f_{12} must be selected in the configuration process and f_2 is a mandatory feature for f_{11} and f_{12} . Then, f_2 which must be selected should be the core feature and it is IACF.

In more in-depth view, the main reason for the occurrence of IACF is that the variant constraints of core features make one of their optional child-features selected in the configuration process. It is precisely that these optional child-features have a common child and this child corresponded to these optional child-features is mandatory. Then this child that we called IACF must be selected. Therefore, a feature is IACF determined by its parents and its parents of the parents.

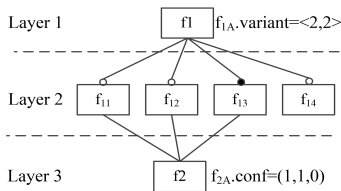


Fig. 6. Example of IACF.

We divide the structure in Fig.6 into three layers from top to bottom. When the following conditions are satisfied, the feature located in third layer must be the core feature.

Theory 1 The feature in first layer is a core feature.

Theory 2 The feature in the second layer is the optional child of the feature in the first layer, and these features all have the same child-feature, which is located in the third layer.

Theory 3 The feature located on the third layer is a mandatory feature relative to the features of the second layer.

Theory 4 Variant constraints of the feature on first layer make the feature on second layer must be selected.

Algorithm 1 get DACF set

Input: S_f

Output: S_{DACF}

```

1: function GETDACF( $S_f$ )
2:    $S_{DACF} \leftarrow \phi$ ;  $queue \leftarrow \phi$ ;  $wasIn \leftarrow \phi$ ;  $f_t \leftarrow \phi$ 
3:   for each  $f \in S_f$  do
4:      $queue.clear()$ ;  $wasIn.clear()$ 
5:      $queue.add(f)$ ;  $wasIn.add(f)$ 
6:     while  $queue \neq \phi$  do
7:        $f_t \leftarrow queue.poll()$ 
8:       if  $f_t \notin S_{cf}$  then
9:          $S_{DACF} \leftarrow f_t$ 
10:      end if
11:      for each  $f_c \in f_t.child$  do
12:        if  $f_c \notin wasIn$  &  $f_c.conf = (f_t, 1)$  &
            $f_c \notin S_{cf}$  then
13:           $wasIn \leftarrow f_c$ ;  $queue \leftarrow f_c$ ;
14:        end if
15:      end for
16:    end while
17:  end for
18:  return  $S_{DACF}$ 
19: end function

```

Next, we introduce the algorithms to extract DACF, DTCF and IRCF, then extract the core features completely.

B. Data Structures

We use the following data structures:

```

Feature{
  name:String
  child:list<Feature>
  parent:list<Feature>
  conf:map<Feature,Boolean>
  var:int[2]{n,m}
  req:list<Feature>
}

```

Where: *name* identifies a feature, *child* is the features child set, *parent* is the features parent set, *conf* is configuration attribute, *var* is variant constraint, *req* is required features.

C. Algorithm

Algorithm 1 gives a process of extracting DACF. The input is a core feature set S_f and output is S_{DACF} . S_{cf} is a

Algorithm 2 get RTCF set

Input: S_f
Output: S_{RTCF}

```
1: function GETRTCFSET( $S_f$ )
2:    $S_{RTCF} \leftarrow \phi$ 
3:   for each  $f \in S_f$  do
4:     GETRTCF( $f$ )
5:   end for
6:   return  $S_{RTCF}$ 
7: end function
8: function GETRTCF( $f$ )
9:   if  $f.req = \phi$  then
10:    return
11:   end if
12:   for each  $f_r \in f.req$  do
13:     if  $f_r \in S_{cf}$  then
14:       continue
15:     end if
16:      $S_{RTCF} \leftarrow f_r; \text{GETRTCF}(f_r)$ 
17:   end for
18: end function
```

global collection that holds all core features. The algorithm utilizes the breadth-first traversal of the graph, and sequentially searches for the mandatory child-features path starting from the feature f in the S_f . If f has multiple parents which are core features, as long as f is a mandatory feature relative to one of these parents, f is the core feature (line 12). Because of the mandatory path is formed.

Algorithm 2 gives the process of extracting the RTCF. Similarly, a core feature set is used as input, and the required transfer path starting from feature f in S_f is searched recursively, and all the features in the path are output. If a required feature of f is already the core feature, we ignore this feature to avoid repeated access (line 13-15).

Algorithm 3 gives the process of extracting IACF. As mentioned above, IACF must be a multi-parent features f_{mp} . So we take the multi-parent features set S_{mp} as input. The S_{mp} can be obtained by traversing the feature model once time (similar to algorithm 1), which is not discussed here. The feature f_{mp} is that IRCF determined by the attributes and structure of its parent-feature set S_p and its parents of parents feature set S_{pp} . These features must satisfy the above four conditions. Firstly, the parent relationship is used to find the f_p which needs to satisfy **Theory2** and **Theory3** (lines 8-12). The f_p that satisfies the condition is stored in the S_p . In this set, we search for all f_p 's parent-feature f_{pp} which is core feature to satisfy **Theory1**, and store the valid f_{pp} in S_{pp} (lines 13-19). To satisfy **Theory4**, traverse the S_{pp} to get the number of all optional child-features N_{aoc} of f_{pp} , the number of child-features $N_{ocisp} \in S_p$, and the variant constraint value $[n, m]$ (lines 20-27). If $n \geq N_{aoc} - N_{ocisp} + 1$, it indicates that f_p must be selected in the S_p . Therefore f is the core feature.

Algorithm 4 introduces the main flow of extracting core feature set. The root feature is used as an input and output is

Algorithm 3 get IACF Set

Input: S_{mp}
Output: S_{IACF}

```
1: function GETIACF( $S_{mp}$ )
2:    $S_{IACF} \leftarrow \phi$ 
3:   for each  $f \in S_{mp}$  do
4:      $S_p \leftarrow \phi; S_{pp} \leftarrow \phi$ 
5:     if  $f \in S_{cf}$  then
6:       continue
7:     end if
8:     for each  $f_p \in f.parent$  do
9:       if  $f.conf = (f_p, 1) \& f_p \notin S_{cf}$  then
10:         $S_p \leftarrow f_p$ 
11:       end if
12:     end for
13:     for each  $f_p \in S_p$  do
14:       for each  $f_{pp} \in f_p.parent$  do
15:         if  $f_{pp} \in S_{cf}$  then
16:            $S_{pp} \leftarrow f_{pp}$ 
17:         end if
18:       end for
19:     end for
20:     for each  $f_{pp} \in S_{pp}$  do
21:        $n_{aoc} \leftarrow f_{pp}.optChildNum$ 
22:        $variant \leftarrow f_{pp}.var[0]; n_{ocisp} \leftarrow 0$ 
23:       for each  $f_c \in f_{pp}.child$  do
24:         if  $f_c \in S_p$  then
25:            $n_{ocisp}++$ 
26:         end if
27:       end for
28:       if  $variant \geq (n_{aoc} - n_{ocisp} + 1)$  then
29:          $S_{IACF} \leftarrow f$ 
30:         break
31:       end if
32:     end for
33:   end for
34:   return  $S_{IACF}$ 
35: end function
```

core feature set S_{cf} . When the core features are acquired each time using the above three algorithms, the newly discovered core features may cause other features associated with them to be transformed into core features. For example, a new core feature's mandatory child-feature or required feature is also the core feature. Therefore, we reuse the three algorithms to continuously synchronize the new core features to the S_{cf} until there are no new core feature. The set S_{temp} is used to temporarily store the core features acquired by a certain algorithm and use it as input to other algorithms. The set S_{mp} stores multi-parent features. In the inner loop, we alternately use algorithms 1 and 2 to obtain core features until there are no new ones (lines 4-17). In the outer loop, we judge whether there are new IACF in the multi-parent features. If new core features appear, it is necessary to go through the inner loop. If it does not appear, it means that the core feature set has been

Algorithm 4 get Core Feature Set

Input: f_{root} **Output:** S_{cf}

```
1: function GETCOREFEATURE( $f_{root}$ )
2:    $S_{temp} \leftarrow GETDACF(f_{root})$ 
3:    $S_{mp} \leftarrow FINDMULTIPARENT(f_{root}); S_{cf} \leftarrow \phi$ 
4:   while (true) do
5:     while (true) do
6:       if  $S_{temp} = \phi$  then
7:         break
8:       end if
9:        $S_{cf} \leftarrow S_{temp} \cup S_{cf}$ 
10:       $S_{temp} \leftarrow GETRTCFSET(S_{temp})$ 
11:      if  $S_{temp} = \phi$  then
12:        break
13:      else
14:         $S_{cf} \leftarrow S_{temp} \cup S_{cf}$ 
15:         $S_{temp} \leftarrow GETDACF(S_{temp})$ 
16:      end if
17:    end while
18:     $S_{temp} \leftarrow GETIACF(S_{mp})$ 
19:    if  $S_{temp} = \phi$  then
20:      return  $S_{cf}$ 
21:    else
22:       $S_{temp} \leftarrow S_{temp} \cup GETRTCFSET(S_{temp})$ 
23:       $S_{cf} \leftarrow S_{temp} \cup S_{cf}$ 
24:       $S_{temp} \leftarrow GETDACF(S_{temp})$ 
25:    end if
26:  end while
27: end function
```

obtained completely.

V. CASE STUDY

We applied the above FM to the COPD home care system. As shown in Fig.7, it is part of the FM diagram of our product line, which takes the core feature *sign monitoring* as the root feature and covers the examples above. We use feature attributes and feature relations to describe the whole model, and apply the algorithm to extract the core features of the examples. According to the algorithm, $\{f1, f2, f3, f4, f9, f11, f19\}$ is the DACF. $\{f14\}$ is the RTCF, and $\{f15\}$ is the IACF. The final domain model has 336 features, including 264 service features, 64 technical features, 8 hardware features and 47 constraints. We implement the core feature extraction method, and the FM is transformed into a file representation in XML format. After applying the algorithm, 101 core features are obtained, and the proportion of core features to total features is 30.05%. Among the core features, 91 are DACF, 8 are RTCF, and 2 are IACF.

VI. CONCLUDE

In this paper, an enhanced feature meta-model is proposed to solve the ambiguous semantics, enrich the expressive ability of variable requirements, and provide a hierarchical way to

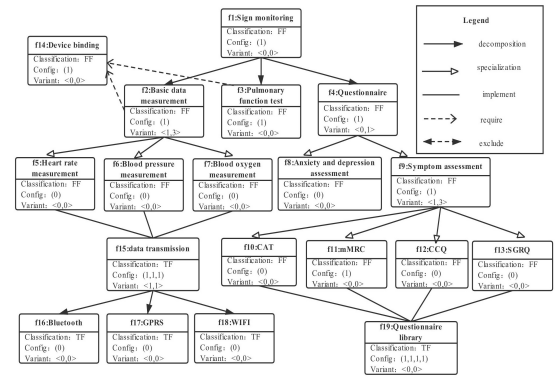


Fig. 7. Feature model fragment of COPD home care system.

construct feature model. Then, a heuristic strategy is proposed to identify and extract the core feature set on the meta-model. We implemented and applied this method to the product line of COPD family care system. The next step is to implement a visualization tool for our modelling method.

REFERENCES

- [1] K. Pohl, G. Bockle, and F. J. V. D. Linden, "Software product line engineering: Foundations, principles, and techniques," *Proceedings of the First Intl Workshop on Formal Methods in Software Product Line Engineering*, vol. 49, no. 12, pp. 29–32, 2005.
- [2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, Tech. Rep., 1990.
- [3] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "Form: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, no. 1, p. 143, 1998.
- [4] K. Czarnecki, S. Helsen, and U. Eisencker, "Formalizing cardinality-based feature models and their specialization," *Software process: Improvement and practice*, vol. 10, no. 1, pp. 7–29, 2005.
- [5] T. Tenório, D. Dermeval, and I. I. Bittencourt, "On the use of ontology for dynamic reconfiguring software product line products," in *Proceedings of the ninth international conference on software engineering advances*, 2014, pp. 545–550.
- [6] D. Fey, R. Fajta, and A. Boros, "Feature modeling: A meta-model to enhance usability and usefulness," in *International Conference on Software Product Lines*. Springer, 2002, pp. 198–216.
- [7] T. Asikainen, T. Mannisto, and T. Soinen, "A unified conceptual foundation for feature modelling," in *10th International Software Product Line Conference (SPLC'06)*. IEEE, 2006, pp. 31–40.
- [8] J. Peña, M. G. Hinchey, A. Ruiz-Cortés, and P. Trinidad, "Building the core architecture of a nasa multiagent system product line," in *International Workshop on Agent-Oriented Software Engineering*. Springer, 2006, pp. 208–224.
- [9] D. Benavides, P. Trinidad, and A. R. Cortés, "Using constraint programming to reason on feature models," in *SEKE*, 2005, pp. 677–682.
- [10] H. Perez-Morago, R. Heradio, D. Fernandez-Amoros, R. Bean, and C. Cerrada, "Efficient identification of core and dead features in variability models," *IEEE Access*, vol. 3, pp. 2333–2340, 2015.
- [11] J. Van Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in *Proceedings Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2001, pp. 45–54.
- [12] A. Abbas, I. F. Siddiqui, S. U.-J. Lee, and A. K. Bashir, "Binary pattern for nested cardinality constraints for software product line of iot-based feature models," *IEEE Access*, vol. 5, pp. 3971–3980, 2017.
- [13] C. F. Vogelmeier, G. J. Criner, F. J. Martinez, A. Anzueto, P. J. Barnes, J. Bourbeau, et al., "Global strategy for the diagnosis, management, and prevention of chronic obstructive lung disease 2017 report, gold executive summary," *American journal of respiratory and critical care medicine*, vol. 195, no. 5, pp. 557–582, 2017.

SSLDoc: Automatically Diagnosing Incorrect SSL API Usages in C Programs

Zuxing Gu, Jiecheng Wu, Chi Li, Min Zhou, Ming Gu
School of Software Engineering, Tsinghua University, Beijing, China, 100084

Abstract—Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols provide a reliable communication channel between applications over the Internet. Implementations of these protocols (e.g., OpenSSL and GnuTLS) publish well-format documentation and examples online to guide the usage of SSL/TLS APIs. However, incorrect usages have caused many severe vulnerabilities (e.g., privilege escalation, denial of service, man-in-the-middle attack, etc.) in recent years. In this paper, we introduce SSLDoc to diagnose incorrect SSL API usages in real-world C programs automatically. The key insight behind SSLDoc is a constraint-directed static analysis technique powered by domain-specific usage patterns that we learn from real-world vulnerabilities and bug-fix-related patches. We have instantiated SSLDoc for OpenSSL APIs and applied it to large-scale open-source programs. SSLDoc found 45 previously unknown security-sensitive bugs in OpenSSL implementation and applications in Ubuntu. We created and submitted issues for all of them. Up to now, 35 have been confirmed by the corresponding development communities and 27 have been fixed in master branch.

Index Terms—SSL, API usage validation, static analysis, bug detection

I. INTRODUCTION

Secure Socket Layer (SSL) and Transport Layer Security (TLS) are the most widely deployed protocols in security-sensitive software. They provide a confidential and authentic end-to-end communication mechanism against an active, man-in-the-middle attacker. The details of these protocols are complicated and involves many steps to set up and validate certificate authority [1], [2]. Therefore, client programs usually rely on SSL libraries such as OpenSSL [3] and GnuTLS [4], which encapsulate the internal details and diverse kinds of cryptography algorithms into APIs with well-format documentation and examples. However, correct usage of SSL APIs is required to satisfy certain constraints, such as call conditions or call orders. Violations of these constraints will lead to software bugs and more critically, can have severe security implications. For example, missing error status code validation of SSL APIs will cause a denial of service by remote attackers (CVE-2016-2182 [5]), and broken SSL certificate validation will result in man-in-the-middle attacks [6]. A recent study show that SSL certificate validation is completely broken in many security-critical applications and libraries [7].

Many different tools, techniques and methodologies have been proposed to address the above problems. Clark et al. [8] present a comprehensive survey of SSL issues to enhance the certificate infrastructure used in practice. Brubaker et al. [9]

systematically test the correctness of the certificate validation logic in SSL/TLS implementations. However, they focus on SSL implementation and require considerable manual efforts to prepare a test environment.

To automatically detect incorrect usages of SSL APIs in client programs, static analysis has long prevailed as one of the most promising techniques [10]. For example, He et al. [11] design and implement SSLINT, a scalable static analysis tool to match a program dependence graph with a handcrafted, precise signature modeling the correct logic usage of SSL APIs. Although SSLINT is capable of detecting incorrect usages in practice, it is hard to apply to APIs without predefined signatures and produces many false positives and false negatives due to imprecise static analysis (e.g., flow-insensitive and context-insensitive). Yun et al. [12] present APISan for incorrect API usages of causal relation and semantic relation on arguments with security implications by leveraging the strength of static analysis (such as control dependency analysis) and code mining (such as frequent sub-itemsets mining algorithm). It provides accurate detection and can be applied to scale real-world system programs. However, a challenge for such tools is insufficient data to train models, which is particularly severe for SSL APIs in client programs.

In this paper, we aim at augmenting current detection capability of incorrect SSL API usage for large-scale C programs. The key insight is a constraint-directed static analysis technique powered by domain-specific usage patterns. To understand the root causes of incorrect usages of SSL APIs, we begin with a preliminary investigation of real-world vulnerabilities to summarize generic incorrect usage patterns. Leveraging this knowledge, we design and implement SSLDoc, a static analysis detector employing under-constrained symbolic execution [13] to generate abstract symbolic traces with rich semantics and detect incorrect usages. In this way, SSLDoc can precisely conduct a flow-, control- and context-sensitive analysis inter-procedurally (i.e., capable of capturing temporal sequencing of API calls, path constraints, and data flows between parameters and return values in or across procedures).

To evaluate SSLDoc in practice, we instantiated it with OpenSSL APIs and applied it to more than half million lines of source code, including OpenSSL implementation and 15 applications in Ubuntu. The result shows that SSLDoc discovers 45 previously unknown security-sensitive incorrect SSL API usages. We reported our findings to developers and received 35 confirmations, out of which 27 have been fixed in multiple branches. Moreover, we share the lessons

learned from bug detecting, issue reporting and discussions with developers.

In summary, our paper makes the following contributions:

- We design and implement SSLDoc, a static analysis tool to augment current detection capability of incorrect SSL API usage for large-scale C programs.
- We instantiate SSLDoc with OpenSSL APIs and apply it to real-world programs. It discovers 45 previously unknown incorrect SSL API usages, out of which 35 have been confirmed by developers.
- We share the lessons learned from bug detecting, issue reporting and discussions with developers in practice. We hope our findings can motivate more researcher to combat incorrect SSL API usages.

The rest of this paper is organized as follows. Section II provides motivating examples of our work. Section III presents the design of SSLDoc, followed by an evaluation in Section IV. We share the lessons learned in Section V and discuss related work in Section VI and conclude in Section VII.

II. MOTIVATING EXAMPLE

Instead of implementing SSL themselves, Client programs usually rely on APIs of SSL libraries such as OpenSSL and GnuTLS as well as higher-level data-transport libraries such as Curl [14]. While APIs encapsulate the details, they also expose rich semantic constraints. Violations of these constraints, in turn, lead to serious security problems.

To better understand incorrect SSL API usage patterns and how developers fix them in practice, we manually studied four years’ (from 2013 to 2017) CVE entries related to API usage bugs in National Vulnerability Database¹. They are extracted through approximate keywords matching (e.g., “OpenSSL API usage” and “incorrect SSL usage”) and contain concrete patches to fix the bugs. We investigate both the CVE description messages and patches, and identify two generic incorrect usage patterns as shown in Figure 1:

- **Certificate Validation.** SSL libraries encapsulate the core functionality of protocols and export APIs to utilize the implementation. However, the client needs to validate all kinds of certificates in applications. Missing validations might allow attackers to cause a denial of service or man-in-the-middle via an invalid one. Figure 1a shows an example of such vulnerabilities reported in CVE-2015-0288 [15]. Function `X509_get_pubkey()`² attempts to decode the public key for `x`. If an error occurs, it will return NULL. In function `X509_to_X509_REQ()`, the return value `pktmp` is used without checking the error code, which results in a NULL Pointer Dereference bug. Beyond null pointer checking, SSL libraries use various error protocols in practice (e.g., 0 or negative for errors in OpenSSL, but -1 to -403 in GnuTLS).
- **Causal Function Calling** SSL libraries allocate memory resources for cryptography algorithm computing, which

```

1 Location: OpenSSL/crypto/x509/x509_req.c: 70
2 X509_REQ *X509_to_X509_REQ(...) {
3     [...]
4     pktmp = X509_get_pubkey(x);
5     // missing certificate validation of pktmp
6     + if (pktmp == NULL)
7     +     goto err;
8     i = X509_REQ_set_pubkey(ret, pktmp);
9     EVP_PKEY_free(pktmp);
10    [...]
11    }
12    ===== Correct Usage =====
13    Location: /crypto/x509/x509_cmp.c: 390
14    int X509_chain_check_suitesb(...) {
15        [...]
16        pk = X509_get_pubkey(x);
17        rv = check_suite_b(pk, -1, &tfllags);
18        [...]
19    }
20    static int check_suite_b(EVP_PKEY *pkey, ...) {
21        [...]
22        // ensure pkey not NULL
23        if (pkey && ...)
24            [...] // error handling
25    }

```

(a) Incorrect usage for missing certificate validation reported in CVE-2015-0288 [15].

```

1 Location: OpenSSL/ssl/tl_lib.c: 3567
2 static int tls_decrypt_ticket(...) {
3     EVP_CIPHER_CTX ctx;
4     [...]
5     EVP_CIPHER_CTX_init(&ctx);
6     [...] // Check HMAC of encrypted ticket
7     if (CRYPTO_memcmp(ticket_hmac, etick + eticklen, mlen))
8     + { EVP_CIPHER_CTX_cleanup(&ctx);
9       return 2;
10    }
11    [...]
12    sdec = OPENSSL_malloc(eticklen);
13    if (!sdec) {
14        EVP_CIPHER_CTX_cleanup(&ctx);
15        return -1;
16    }
17    EVP_CIPHER_CTX_cleanup(&ctx);
18    [...]

```

(b) Incorrect usage for missing releasing resource reported in CVE-2014-3567 [16].

Fig. 1: Motivating examples of incorrect SSL API usages.

should release after their lifecycle by invoking a causal function calling. Violations of such causal relation (i.e., a-b pattern) will cause a denial of service (memory consumption) via an intentionally crafted input by remote attackers. For example, `EVP_CIPHER_CTX_cleanup()`³ clears all information from a cipher context `ctx` and free up any allocated memory associated with it. However, missing invoking it along the error handling path of `tls_decrypt_ticket()` will be exploited by a crafted session ticket that triggers an integrity-check failure as shown in Figure 1b.

Detection of the above bugs is not trivial. It requires a wide spectrum of semantics instead of simply syntactic matching. For example, function invocation of `X509_get_pubkey()` at Line 16 in Figure 1a is correct, because `check_suite_b()` validates the first parameter to ensure that `pkey` is not NULL. To filter out such instance, it

¹<http://cve.mitre.org/>

²https://www.openssl.org/docs/manmaster/man3/X509_get_pubkey.html

³https://www.openssl.org/docs/man1.0.2/crypto/EVP_CIPHER_CTX_cleanup.html

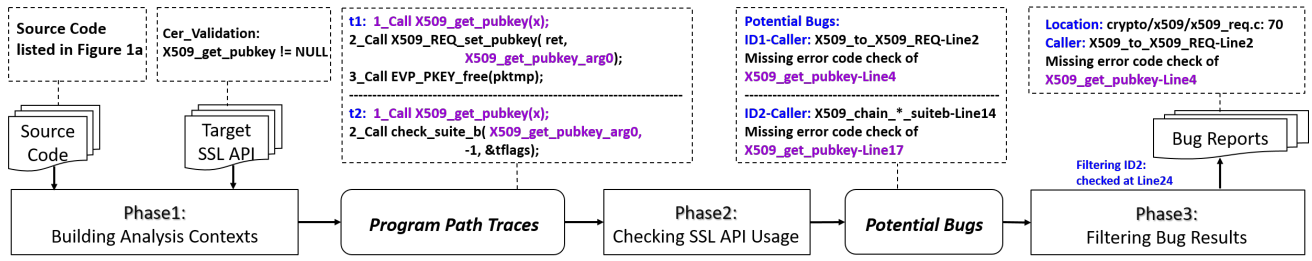


Fig. 2: Overview of SSLDoc's workflow.

demands a flow- and context-sensitive semantic analysis inter-procedurally. Moreover, path-sensitive analysis also should be taken into consideration to check memory leak along different error handling paths (e.g., Line 14 and 17 in Figure 1b).

III. APPROACH

In this section, we introduce SSLDoc, a static analysis tool to augment current SSL API usage detection capability for large-scale C programs. We first present a brief overview of our approach with an example of Figure 1a and elaborate each step of bug detection in the following parts.

As shown in Figure 2, SSLDoc takes the source code and target APIs as input and generates bug reports with concrete locations and reasons as output. Bug detection consists of three basic steps. (1) In **Phase-1**, the analysis context is built by constructing the control flow graph and creating *program path traces* for each target API by employing under-constrained symbolic execution. In this example, two traces, t_1 and t_2 , are generated, as shown in the box above **Program Path Traces**. In this way, SSLDoc can successfully capture the usage context of `X509_get_pubkey()`, `EVP_PKEY_free()` and those in between. (2) In **Phase-2**, SSLDoc employs the *traces* to detect violations of API usages as potential bugs. For example, two API-misuse instances of `X509_get_pubkey()` are found for missing certificate validations labeled as *Potential Bugs*. (3) In **Phase-3**, SSLDoc improves the detection precision by leveraging inter-procedural semantics and usage statistics. Then, the second misuse is filtered out for the check conducted in the `X509_to_X509_REQ()` at Line-24. We discuss the details of our approach as follow.

A. Building Analysis Contexts

SSLDoc performs symbolic execution to generate program path traces that capture rich semantic information for each

target API. In Figure 3, we illustrate the workflow for building analysis context, which consists of three steps. First, SSLDoc parses the source code and builds a control flow graph (CFG). Then, for each target API f , we select analysis entries as target API call sites by labeling the callers C , which invokes f . Next, for each caller $c \in C$, symbolic execution is employed to generate a series of program path traces T with rich semantics of usages of f while traversing the CFG.

We use \mathbb{N}, \mathbb{Z} to denote the set of non-negative and all integers, respectively. In Figure 4, we formally describe the structure of program path traces computed by SSLDoc, where $id \in \mathbb{N}, n \in \mathbb{N}, z \in \mathbb{Z}$ and ap is short for Accesspath [17] to represent memory locations in the form of regular expressions. Each trace t consists of a sequence of actions a^+ with a value map V . In particular, **Assume** action is used to capture path-sensitive semantics. All the actions are labeled while traversing CFG to support flow-sensitive analysis. V records the semantics from a symbolic variable sv to a concrete value cv . A symbolic variable is defined by an action labeled by id and the index n . For example, $id_f_arg_i$ denotes the i^{th} parameter of f called in the id^{th} action. In this way, we can capture the invocation context semantics. We use f_arg_0 to represent the return value of f and arg_0 for the symbolic variable returned by the caller c of f in **Return** action. Therefore, our program path trace is capable of capturing the flow-, context- and path-sensitive semantics.

In Figure 5, We illustrate three traces of the example code listed in Figure 1a. t_1 and t_3 are original code snippets, and t_2 is with the bug-fix patch. All traces start from the action calling `X_509_get_pubkey()`. Then, t_1 directly passes the return value `1_X509_get_pubkey_arg_0` into `X509_REQ_set_pubkey()` without certificate validation. By contrast, t_2 validates the return value immediately. Even though t_3 passes it into `check_suite_b()` without valida-

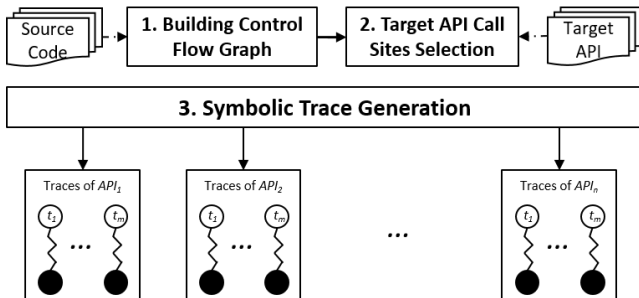


Fig. 3: Workflow of building analysis contexts.

(Traces) $T ::= t$
 (trace) $t ::= (id_a)^+; V$
 (action) $a ::= \text{Assume}(exp) \mid \text{Call } f(sv^*) \mid \text{Return}(sv)$
 (expression) $exp ::= sv1 \text{ cmpop } sv2$
 (value map) $V ::= sv \rightarrow cv$
 (symbolic variable) $sv ::= (id_f_arg_n)$
 (compare operator) $cmpop ::= != \mid == \mid >= \mid > \mid <= \mid <$
 (concrete value) $cv ::= z \mid ap \mid \text{NULL}$
 (function) $f \in \mathbb{F}$

Fig. 4: Abstract syntax of program path traces.


```

t1 : 1 _Call X509_get_pubkey(_);
      2 _Call X509_REQ_set_pubkey(_, 1_X509_get_pubkey_arg_0);
t2 : 1 _Call X509_get_pubkey(_);
      2 _Assume(1_X509_get_pubkey_arg_0 != NULL);
      3 _Call X509_REQ_set_pubkey(_, 1_X509_get_pubkey_arg_0);
t3 : 1 _Call X509_get_pubkey(_);
      2 _Call check_suite_b(1_X509_get_pubkey_arg_0, _, _);
      2 _Assume(1_X509_get_pubkey_arg_0 != NULL);

```

Fig. 5: Program path traces of the code in Figure 1a, where we use “_” to represent values irrelevant.

tion, `check_suite_b()` checks the first parameter at Line 24 in Figure 1a.

Similar to the traditional analysis, the key challenge of building such path traces in large and complex programs is to overcome the path-explosion problem. We make two design decisions to achieve scalability without sacrificing substantial accuracy. (1) Limiting inter-procedural analysis. SSLDoc performs symbolic execution intre-procedurally for each caller c of the target API f at most two depth (i.e. we track c and callees of c). We refine the bug detection results by a filtering phase with the deeper inter-procedural semantics presented in Section III-C. (2) Unrolling loops. SSLDoc unrolls each loop only once to reduce the number of paths explored. While this restriction can limit the accuracy of the semantic computation, it does not noticeably affect the accuracy of SSL API bug detection for only a small number of usages related to loop variables.

B. Checking API misuses

In the checking phase, SSLDoc employs target APIs and the program path traces T to detect bugs. To configure usage pattern of SSL APIs, we provide each target f with a usage pattern type $f.\mathcal{T}$, which can be certificate validation with a predicate \mathcal{P} (e.g., `X509_get_pubkey != NULL`) and causal function calling with a usage pattern \mathcal{C} (e.g., `EVP_CIPHER_CTX_init(arg1) → EVP_CIPHER_CTX_cleanup(arg1)`, where `arg1` labels the target memory object). We illustrate our detecting algorithm in Algorithm 1. First, we extract all the target functions into $APISet$. For each API f , we detect API-misuse bugs along the traces T' that invoke f . Then, for each trace t in T' , we validate whether the usage pattern in $f.\mathcal{T}$ are satisfied along t . If a path fails, SSLDoc labels the call site of f along this t as a potential bug. To check the satisfaction of certificate validation along t , we compute the satisfiability of \mathcal{P} . That is, whether there is an **Assume** action to ensure \mathcal{P} . For usage pattern \mathcal{C} , we match with **Call** actions, which satisfy the target memory object constraint. If any of the constraints fail to match, SSLDoc reports a bug. For example, t_1 in Figure 5 fails to ensure the certificate validation `X509_get_pubkey != NULL`, which may result in a null pointer dereference bug.

C. Filtering Bug Reports

To achieve the scalability required to support real-world programs, we employ a limiting inter-procedural strategy to ad-

Algorithm 1 Algorithm for checking incorrect SSL API usage

Input: program path traces T , target APIs \mathbb{F}
Output: bug report R

```

1:  $R \leftarrow \emptyset$ 
2:  $APISet \leftarrow \text{extractTargetAPISet}(\mathbb{F})$ 
3: for each API  $f \in APISet$  do
4:    $T' \leftarrow \text{extractPathTraces}(f, T)$ 
5:   for each trace  $t \in T'$  do
6:      $result \leftarrow \text{satisfy}(t, f.\mathcal{T})$ 
7:     if ( $\neg result$ ) then
8:        $R \leftarrow \text{addBug}(t, f)$ 
9:     end if
10:  end for
11: end for
12: return  $R$ 

```

dress the path-explosion problem. The strategy generates false positives when a usage cross more than two functions. However, developers dislike using tools with low precision [18]. Therefore, we apply deeper inter-procedural semantics and rank the final results according to usage statistics.

First, we conduct semantic-based filtering. We attempt to infer semantics across functions. For the missing validation of certificate x , we further check the functions which directly receive x as a parameter. If these functions contain sanity check against x , we filter it out. For causal function calling pattern as $a \rightarrow b$, if the target memory object of a is directly assigned to the parameter of the caller c of a or returned by c , we check whether callers \mathbb{C} of c invoke b . We filter out the cases that contain function invocation of b .

Then, we conduct a usage-based ranking. Basically, we compute the number of correct/incorrect usage traces respectively, ranks bug reports in decreasing order of their likelihood of being bugs as $\mathcal{H}(f) = \frac{\# \text{of correct usage traces of } f}{\# \text{of incorrect uage traces of } f}$. The highest likelihood value indicates that more correct usages occur and the violations are less. Therefore, the violations are highly buggy. Note that, we use the trace number instead of call site number, because of the observation that many bugs occur along path-branches with different context semantics. However, we have to specially treat when $\mathcal{H}(f)$ is 0, because it indicates that all the usages are buggy. The preliminary experiment results show that it frequently occurs in small programs which invoke SSL APIs only once or twice.

D. Implementation

SSLDoc is built in Java language. We preprocess the source code into LLVM-IR 3.9⁴, which provides a typed, static single assignment (SSA) and well-suited low-level language. Then, we parse the LLVM-IR by javacpp⁵ and construct an extended control flow graph, which classifies the edges into control edges for semantic computation and summary edges to provide a mechanism to support large-scale programs. We have integrated part of OpenSSL APIs with SSLDoc and provide an interface to extend our analysis in a human-readable format named Yaml⁶.

⁴<http://releases.llvm.org/3.9.0/docs/ReleaseNotes.html>

⁵<https://github.com/bytedeco/javacpp>

⁶<http://yaml.org/>

IV. EVALUATION

In this section, we describe our results from incorrect SSL API usage detection on large-scale open-source programs using SSLDoc. We begin by providing the experimental setup. Then we present the security-sensitive bugs we found and concluding with lessons we learned.

a) *Experimental Setup:* We applied SSLDoc to find incorrect SSL API usages in OpenSSL implementation as well as applications using OpenSSL library in Ubuntu 16.04. Target applications are selected by search dependence attributes using package management command line “apt-cache rdepends libssl1.0.0”. In total, we found more than 1200 packages using this library and selected 15 packages which are open-source on Github and ongoing development. For all the 16 programs, we detect incorrect usages in the latest stable versions. Then, we use GNU cflow⁸ to extract target SSL APIs invoked in the applications and create usage pattern mentioned in Section II according to the user manual of OpenSSL⁹. In total, 136 different SSL APIs are integrated with SSLDoc. We ran SSLDoc on Ubuntu 16.04 LTS (64-bit) with a Core i5- 4590@3.30 GHz Intel processor and 16 GB memory.

b) *Result:* Overall, SSLDoc detected 45 previously unknown security-sensitive incorrect SSL API usages as listed in Table I. We tried our best to understand the context and created issues for all the bugs to the developers of each program. Up to now, 32 of the new bugs have been confirmed by the developers and 27 have been fixed in the master branch.

For example, in Figure 6 we present a bug caused by incorrect validation of connect status in dma, a small Mail Transport Agent, which is fixed at 12 hours after we submitted the bug report with bug description and explanation of bug traces. Function `SSL_connect()` initiates the SSL handshake with a server. It returns 0 and negative integers to indicate SSL handshake is not successful. However, the status validation in `dma/crypto.c` only checked against negative integers, which may cause a man-in-the-middle attack leading to leakage of user credentials and emails messages.

V. DISCUSSION

While investigating the bug reports generated by SSLDoc, we find several intricate bugs and gain useful experience in the bug reporting process with open-source developers. We share our following experience. **(1) Incorrect SSL API usages are not corner cases.** In total, we find 45 previously unknown incorrect usages. However, OpenSSL library has provided well-format documentation and examples to guide correct usages. These bugs may result from the lack of a bug information sharing mechanism and the lack of API usage constraints among client software developers. We believe that bug fixing is an essential activity during the entire life cycle of software development. Automatic bug-finding tools,

TABLE I: Previously unknown incorrect SSL API usages detected by SSLDoc

Index	Program	Issue ID	Target API	Status
1	OpenSSL 1.1.1-pre8	6567	RAND_bytes	✓✓
2		6568	ASN1_INTEGER_get	✓
3		6569	ASN1_INTEGER_set	✓✓
4		6570	ASN1_object_size	✓
5		6572	BN_set_word	✓✓
6		6573	HMAC_Init_ex	✓
7		6574	EVP_PKEY_get0_DH	✓✓
8		6575	EC_KEY_generate_key	✓
9		6781	EC_GROUP_new_by_curve_name	✓✓
10		6789	ASN1_INTEGER_set	✓✓
11		6820	ASN1_INTEGER_to_BN	✓✓
12		6822	BN_sub	✓✓
13		6973	EVP_MD_CTX_new	✓✓
14		6977	ASN1_INTEGER_set	✓✓
15		6982	OBJ_nid2obj	✓✓
16		6983	BN_sub	✓✓
17		7235	DH_set0_key	✓
18	dma	59	SSL_connect	✓✓
19	exim	2316	X509_NAME_oneline	✓✓
20		2317	SSL_CTX_set_cipher_list	✓✓
21	hexchat	2244	BN_set_word	✓
22		2245	DH_set0_key	P
23	httplib	41	SSL_CTX_new	✓
24	ipmitool	37	MD2_Init	✓
25	open-vm-tools	291	SSL_CTX_set_cipher_list	✓✓
26		292	X509_STORE_CTX_get_current_cert	✓✓
27	irssi	943	SSL_get_peer_certificate	P
28		944	BIO_read	P
29	keepalive	1003	SSL_CTX_new	✓✓
30		1004	SSL_new	✓✓
31	thc-ipv6	28	BN_new	✓✓
32		29	BN_set_word	✓✓
33	FreeRADIUS	2309	BIO_new	✓✓
34		2310	i2a_ASN1_OBJECT	✓✓
35	trafficserver	4292	SSL_CTX_new	P
36		4293	SSL_new	P
37		4294	SSL_write	P
38	tinc	205	BN_hex2bn	✓✓
39		306	RAND_load_file	✓✓
40	sslsplit	224	SSL_CTX_use_certificate	✓✓
41		225	SSL_CTX_use_PrivateKey	✓✓
42	rdesktop	280	BN_bin2bn	P
43		281	BN_mod_exp	P
44	proxytunnel	36	SSL_connect	P
45		37	SSL_new	P

✓✓ is fixed, ✓ is confirmed without a patch, and P is waiting developer response.

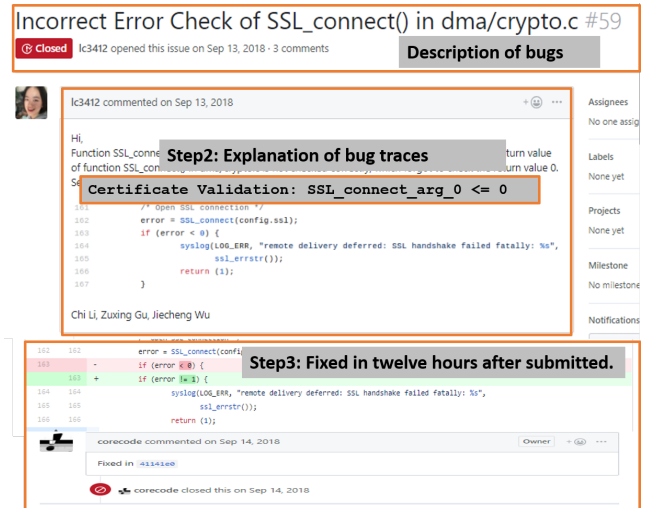


Fig. 6: Screenshot of a bug caused by incorrect validation of `SSL_connect()` status in `dma`, which is fixed at 12 hours.

⁷In Ubuntu16.04 OpenSSL library is listed as libssl1.0.0.

⁸<http://www.gnu.org/software/cflow/>

⁹<https://www.openssl.org/docs/manmaster/man3/>

such as SSLDoc, with large-scale analysis capability can be integrated into the development cycle. In addition, SSLDoc can be customized to incrementally address this problem.

(2) Accelerating manual auditing. SSL API usages usually have similar behavior patterns. For example, many types of vulnerabilities result from insufficient validation of input or missing certificate validations. However, discovering all the missing checks by human is tedious and time-consuming. Automatic tools can efficiently accelerate the manual auditing with differences extracted as good usages and bad usages. For example, two of the API misuses were fixed within 12 hours after we created the issues with possible fixing patches, as shown in Figure 6. **(3) Intentional choices.** We also find that many incorrect usages are not mistakes but intentional choices. Many error status code checks of return values are ignored by developers. During the bug reporting process with the OpenSSL developers, we learned that they intentionally ignore some error code checks for performance considerations or due to the lack of an error handling mechanism in C¹⁰.

VI. RELATED WORK

A few works in the past have analyzed application vulnerabilities due to improper usage of SSL/TLS. Georgiev et al. [7] employ dynamic analysis to conduct MITM attacks and demonstrate that SSL certificate validation is completely broken due to badly designed APIs of SSL implementations. Later, Clark et al. [8] present a comprehensive survey of SSL security and Brubaker et al. [9] apply Frankencerts, a smart fuzzer to test SSL/TLS certificate validation code in implementation. He et al. [11] develop SSLINT, a scalable, automated, static analysis system for detecting incorrect certificate validation vulnerabilities in client programs with pre-defined API signatures. To automatically infer usage pattern, Yun et al. [12] present APISan to infer correct API usages from source code without manual effort and detect various properties with security implications. Moreover, generic bug detection approaches also can be applied to SSL/TLS API usage, such as static analysis approaches [19], [20] and testing [21]. SSLDoc specifically targets SSL API usages in C programs and complements these works. In addition, our work can be easily extended to other domains.

VII. CONCLUSION

Client programs rely on APIs of libraries implementing SSL/TLS protocols to ensure reliable communications. Incorrect usage of such APIs will cause security-sensitive problems, even severe vulnerabilities. In this paper, we present SSLDoc, a static analysis detector to automatically diagnose incorrect usages of SSL APIs in C programs. We instantiate SSLDoc with APIs of OpenSSL and apply it to large-scale programs. We find 45 previously unknown bugs in OpenSSL implementation and 15 applications in Ubuntu which use SSL APIs, out of which 27 have been fixed. We share the lessons learned from bug detection and discussions with developers to

motivate more researchers and practitioners to combat incorrect SSL API usages.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their helpful feedback. This research is sponsored in part by National Natural Science Foundation of China (Grant No. 61802259, 61402248, 61527812), National Science and Technology Major Project of China (Grant No. 2016ZX01038101), and the National Key Research and Development Program of China (Grant No. 2015BAG14B01-02, 2016QY07X1402).

REFERENCES

- [1] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," Tech. Rep., 2008.
- [2] A. Freier, P. Karlton, and P. Kocher, "The secure sockets layer (ssl) protocol version 3.0," Tech. Rep., 2011.
- [3] "Openssl: cryptography and ssl/tls toolkit." <https://github.com/openssl/openssl>, 2019.
- [4] "Gnutls: a secure communications library implementing the ssl, tls and dtls protocols and technologies around them." <https://gitlab.com/gnutls/gnutls/>, 2019.
- [5] "Cve-2016-2182," <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2182>, 2016.
- [6] "Cve-2016-2113," <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2113>, 2016.
- [7] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The most dangerous code in the world: validating SSL certificates in non-browser software," in *CCS'12, Raleigh, NC, USA, October 16-18, 2012*, 2012, pp. 38–49.
- [8] J. Clark and P. C. van Oorschot, "Sok: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements," in *SP 2013, Berkeley, CA, USA, May 19-22, 2013*, 2013, pp. 511–525.
- [9] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations," in *SP 2014, Berkeley, CA, USA, May 18-21, 2014*, 2014, pp. 114–129.
- [10] A. Delaitre, B. Stivalet, E. Fong, and V. Okun, "Evaluating bug finders - test and measurement of static code analyzers," in *COUFLESS 2015, Florence, Italy, May 23, 2015*, 2015, pp. 14–20.
- [11] B. He, V. Rastogi, Y. Cao, Y. Chen, V. N. Venkatakrishnan, R. Yang, and Z. Zhang, "Vetting SSL usage in applications with SSLINT," in *SP 2015, San Jose, CA, USA, May 17-21, 2015*, 2015, pp. 519–534.
- [12] I. Yun, C. Min, X. Si, Y. Jang, T. Kim, and M. Naik, "Apisn: Sanitizing API usages through semantic cross-checking," in *USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, 2016, pp. 363–378.
- [13] D. A. Ramos and D. R. Engler, "Under-constrained symbolic execution: Correctness checking for real code," in *USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, 2015, pp. 49–64.
- [14] "Curl: A command line tool and library for transferring data with url syntax." <https://github.com/curl/curl>, 2019.
- [15] "Cve-2015-0288," <https://www.cvedetails.com/cve/CVE-2015-0288/>, 2015.
- [16] "Cve-2014-3567," <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3567>, 2015.
- [17] B. Cheng and W. W. Hwu, "Modular interprocedural pointer analysis using access paths: design, implementation, and evaluation," in *PLDI 2000, Vancouver, British Columbia, Canada, June 18-21, 2000*, 2000, pp. 57–69.
- [18] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Gros, A. Kamsky, S. McPeak, and D. R. Engler, "A few billion lines of code later: using static analysis to find bugs in the real world," *Commun. ACM*, vol. 53, no. 2, pp. 66–75, 2010.
- [19] A. Arusoaie, S. Ciobaca, V. Craciun, D. Gavrilit, and D. Lucanu, "A comparison of open-source static analysis tools for vulnerability detection in c/c++ code," in *SYNASC 2017*, 2017, pp. 161–168.
- [20] S. Amann, H. A. Nguyen, S. Nadi, T. N. Nguyen, and M. Mezini, "A systematic evaluation of static api-misuse detectors," *IEEE Transactions on Software Engineering*, pp. 1–1 (Early Access), 2018.
- [21] M. Kassab, J. F. DeFranco, and P. A. Laplante, "Software testing: The state of the practice," *IEEE Software*, vol. 34, pp. 46–52, 2017.

¹⁰<https://github.com/openssl/openssl/issues/6575>

Multi-Location Program Repair Strategies Learned from Successful Experience

Shangwen Wang^{1,3}, Xiaoguang Mao^{1,3}, Nan Niu², Xin Yi^{1,3}, and Anbang Guo^{1,3}

¹National University of Defense Technology, Changsha, China

²University of Cincinnati, Cincinnati, USA

³Hunan Key Laboratory of Software Engineering for Complex Systems, Changsha, China
{wangshangwen13, xgmao, yixin09, guoanbang12}@nudt.edu.cn, niunn@ucmail.uc.edu

Abstract—Automated program repair (APR) has great potential to reduce the effort and time-consumption in software maintenance and becomes a hot topic in software engineering recently with many approaches being proposed. Multi-location program repair has always been a challenge in this field since its complexity in logic and structure. While some approaches do not claim to have the features for solving multi-location bugs, they generate correct patches for these defects in practice. In this paper, we first make an observation on multi-location bugs in Defects4J and divide them into two categories (i.e., *similar* and *relevant multi-location bugs*) based on the repair actions in their patches. We then summarize the situation of multi-location bugs in Defects4J fixed by current tools. We analyze the twenty-two patches generated by current tools and propose two feasible strategies for fixing multi-location bugs, illustrating them through two detailed case studies. At last, preliminary results prove the feasibility of our methods with the repair of two bugs that have never been fixed before. By learning from successful experience in the past, this paper points out possible ways ahead for multi-location program repair.

Keywords—automated program repair; multi-location bugs; case studies

I. INTRODUCTION

Over the years, researchers develop various *Automated Program Repair* (APR) techniques aiming at reducing the onerous burden of fixing bugs. Generally, these automated repair tools can be classified into two categories, i.e., search-based methodology (e.g., GenProg [1] and RSRepair [2]) and semantics-based methodology (e.g., S3 [3] and Angelix [4]). Search-based repair method (also known as generate-and-validate methodology) generate patch candidates by searching within a predefined fault space determined by Fault Location (FL) techniques and then validate these candidates against the provided test-suite. Semantics-based repair methodology, on the contrary, utilizes semantic information generated by symbolic execution and constraint solving to synthesize patches. These state-of-the-art APR tools make great achievements on single-edit program repair.

Multi-location program repair, which refers to fixing multi-location bugs whose human-written patches contain multiple chunks [4, 5], has become a challenge since the rise of APR due to its complexity. Some recent empirical studies have shown the importance of multi-location repair: The study by Sobreira et al. [5] shows that more than 60% of the bugs in Defects4J [6], a well-known dataset containing 395 real bugs collected from six open-source Java projects, need fixing in multiple points and Zhong and Su [7] draw the conclusion that

programmers make at least two repair actions in total to fix more than 70% of bugs. So far, only Angelix and S3 have been reported to possess special features designed for multi-location bugs by capturing the dependence among multiple program locations. However, as the authors stated in [3], semantic-based repair exclusively modifies expressions in conditions or on the right-hand side of assignments, leading to its not so satisfactory performance. Thus, it is an emerging trend for solving multi-location bugs.

While some tools do not claim that they have the abilities to fix multi-location bugs, they generate correct patches for these bugs when being evaluated. For example, the patch in Listing 1 modifies the types of two variables in different locations and it is a typical multi-location defect. Recently, a tool named ACS [8] has reported to fix this bug successfully. This phenomenon motivates our study. In this paper, we analyze why these patches are generated and how they fix the bugs, aiming to provide practical guidance for future research by learning the experience. We first conduct an empirical study on the multi-location bugs in Defects4J and classify them into two categories according to repair actions in their patches. We then investigate the statistics of multi-location bugs from Defects4J that are successfully fixed by the current tools. We analyze the twenty-two patches for multi-location bugs generated by current tools and propose two suggestions for solving this kind of bugs, illustrating them through two detailed case studies. Preliminary results show the practicalities of our suggestions with the repair of two out of eight multi-location bugs in Defects4J that have never been fixed before. Our methods are successful by micro-adjustment of our suggestions with existing tools.

II. EMPIRICAL OBSERVATION

Two patches for multi-location bugs in Defects4J, Lang#35 and Chart#5, are shown in Listings 1 and 2, respectively. We divide multi-location bugs in this dataset into two categories by analyzing the repair actions in the two instances.

In Listing 1, developer changes two statements at different places into a same oracle-throwing statement. Modifications at each edit point share similar actions in syntax and thus this bug is classified into syntax similar multi-location bugs category (*similar multi-location bugs* for short). In this category, the similar modification we talk about may spread over both a single line of statement (Time#3) and a chunk of codes (Chart#14). Such cases are abundant in Defects4J, such as Time#3 where several similar *if* conditional statements are added, Math#49 where an object instantiation is modified in many functions from a same class, Closure#4 where a conditio-

```

- type = Object.class;
3295 + throw new IllegalArgumentException();
- .....
- return (T[]) new Object[] { null };
3574 + throw new IllegalArgumentException();

```

Listing 1. The patch of Lang#35

```

541 if (x == null) {
542     throw new IllegalArgumentException("Null 'x' argument.");
543 }
544 if (this.allowDuplicateXValues) {
545     add(x, y);
546     return null;
547 }
548
549 // if we get to here, we know that duplicate X values are not permitted
550 XYDataItem overwritten = null;
551 int index = indexOf(x);
552 if (index >= 0 && !this.allowDuplicateXValues) {
553     if (index >= 0) {
554         XYDataItem existing = (XYDataItem) this.data.get(index);
555         overwritten = (XYDataItem) existing.clone();

```

Listing 2. The patch of Chart#5

nal expression is modified similarly at two places, etc. Among the 244 multi-location patches in our dataset, this category has 70 instances, occupying 28.69% of the total amount. A small part of these cases (23/70) share exactly the same operations at each edit point, such as Chart#14, adding the same conditional block in four edit points.

The fixing shown in Listing 2 is different. It involves the addition of an *if* conditional statement and corresponding operations from lines 544-547 and the modification of the content of an *if* conditional statement in line 552. These modifications are compact and have great logical correlation in the program structure and we name this type semantic relevant multi-location bugs (**relevant multi-location bugs** for short). The criterion is from Abstract Syntax Tree (AST) level: *if the node under one modification appears in other modified places in this patch or the modified places are sub-nodes of a common node*, then the bug belongs to this type. Another example is the patch of Mockito#2 where developer first uses *Method Definition Addition* repair action from lines 30-34¹ and then operates *Method Call Addition* in line 11. This type is more popular in our dataset, holding 67.62% (165/244) of the total amount. This result is consistent with our perception that modifications performed at multiple locations aiming at solving a bug should have logical correlations, in most cases.

A small proportion of the dataset which contains 9 patches like Time#2 and Mockito#11 shows differences from the above two conditions: modifications at different places in these patches are neither similar nor logical related. We have not proposed any method for this kind of situation in this paper due to its peculiarity.

Note that the classification is based on the features of modifications at different places and it can provide guidance for repairing. *Similar* type bugs have no logical correlation at each location and thus we may fix these places one by one, however, *relevant* type bugs possess logical correlations at each location and it may affect other places when operating in one place. That is why *relevant* type bugs are more difficult to repair and it is proved through the results which we will show in the next section: more *similar multi-location bugs* have been fixed than *relevant multi-location bugs*. Also note that when counting the number of each category, we use a **relevant first** strategy which means if a patch contains both similar and relevant edits, it belongs to the latter. For example, in Math#74, two similar loop chunks are added but there is another modification about the loops, making this patch belong to relevant bug. The reason for this strategy is that if both kinds of

¹ Due to space limitation, some code snippets are not shown. Please check them in <https://github.com/program-repair/defects4j-dissection>.

TABLE I. STATISTICAL RESULTS

Bug ID	T	PM	S	jGP	jK	N	A	ssF	J	HDR	SF
C5	R					✓					
C14	S						✓				
C19	S						✓				
CL115	R							✓			
L10	R									✓	
L27	R		✓								
L35	S						✓				
L41	R		✓								
L50	S		✓								
L60	S		✓								
M4	S						✓				
M22	S									✓	
M35	S		✓				✓				
M61	R						✓				
M71	S		✓								
M79	R		✓					✓			
M90	R						✓				
M93	R						✓				
M98	S		✓								
M99	S						✓				
Similar	11	-	5	-	-	-	6	-	-	1	-
Relevant	9	-	3	-	-	1	3	2	-	1	-
Total	20	-	8	-	-	1	9	2	-	2	-

Column “T” means the type of this bug and R refers to *relevant* type while S refers to *similar* type. “✓” denotes this bug is successfully fixed by the tool. “Similar” and “Relevant” denote the numbers of different types of bugs fixed by each tool and “Total” denotes the total number of bugs fixed by each tool. It is marked with “-” if the tool cannot fix any bug.

operations are needed for fixing a bug, then the difficulty degree is near to repairing a *relevant* bug.

III. SITUATION STATISTICS

In this section, we investigate how the contemporary APR tools would handle the multi-location bugs. We select ten tools which have been evaluated on this dataset: ProbabilisticModel (PM) [9], SimFix (S) [10], jGenProg (jGP) [11], jKali (jK) [11], Nopol (N) [12], ACS (A), ssFix (ssF) [13], JAID (J) [14], HDRRepair (HDR) [15], and SketchFix (SF) [16]. Note that we adopt the experimental results for jGenProg, jKali, and Nopol reported by Martinez et al. [11] and the results of other approaches come from the corresponding research papers. The results are illustrated in Table I where each tool is represented by its acronym.

Generally speaking, 22 valid patches are generated and 20 multi-location bugs are successfully fixed including 11 *similar* type and 9 *relevant* type, among which M35 and M79 are fixed by two tools. There are only five tools being able to fix these bugs (i.e., SimFix, Nopol, ACS, ssFix, and HDRRepair) among which SimFix and ACS repair the most bugs with 8 and 9, respectively. Nopol and ssFix repair 1 and 2 bugs respectively and they can only fix *relevant multi-location bugs* at this moment. HDRRepair fixes one bug for each type.

IV. LESSONS LEARNED AND SUGGESTIONS

In this section, we propose two suggestions learned from successful experience and illustrate them through two detailed case studies.

A. Case Study 1: Patch of Lang#35 Generated by ACS

We list the patch of Lang#35 generated by ACS in Fig. 1. Another modification chunk performed at line 3578 is the same as the code in the figure. ACS is especially designed for synthesizing conditional expressions containing two steps: variable selection and predicate selection. It uses a method named *Oracle-Throwing* to avoid the crash, thus, it can generate patch as shown. However, why it is able to generate two modification chunks still needs further explanation. Note that Lang#35 is a *similar multi-location bug* which means the two modification points have no correlation in program structure. Thus, there may be multiple test cases aiming at testing different places in the program and they all fail. We find that the test suite for this project contains two failing test cases and when executing, ACS uses a *fitness function* which enables it to continue fixing if the repair actions that have been performed reduce the number of failing test cases (ACS does not introduce this feature in its paper, we get this information after connecting with the authors). Previous studies such as GenProg and HDRepair use fitness functions to guide the selection process of candidate patches while ACS exploits the deduction of failing test cases for solving buggy points one by one, bringing a new idea for *similar multi-location bugs*. The main challenge for applying this strategy is the precondition: the test suite must have enough failing test cases to expose the defects and thus we must strengthen the test suite. Test case purification [17], which means recovering the execution of omitted assertions, has the ability to generate more practical test cases and enhance the performance of test suite. EvoSuite is a commonly used tool for automated test suite generation and empirically, it can increase code coverage up to 63% [18]. Thus, if we first purify the test suite and add test cases, leading to an enhanced test suite, and then use this fitness function to repair, we may be able to solve more multi-location bugs. This strategy is suitable for our study subject, Defects4J, since all the projects in this benchmark are open source projects and the original test suites are manually created which means they may not cover all the entities in the code.

```
3300 //ACS's patch begin
3301 + if (element == null){throw new IllegalArgumentException();}
3302 //ACS's patch end
```

Fig. 1. Patch of Lang#35 generated by ACS

```
- // append the value to the list...
+ if (this.autoSort) {
+   if (overwritten!=null) {
+     this.data.add(-index - 1, new XYDataItem(x, y));
```

Fig. 2. Patch of Chart#5 generated by Nopol

program skips this conditional branch and goes to line 566 directly and thus the wrong expression in line 548 does not cause the error in line 564 which means the error is avoided. This strategy is to some extent like **fault tolerance** technique [19] and it is the same principle with ACS not producing human-written patch like Listing 1. We further study the reason for modifying line 563.

Recently, **Error Propagation Chain (EPC)**, which refers to a sequence of statements between program defect and program failure statement, is proposed by Guo et al. [20] to improve the efficiency of fault localization. We check their experiment results and find that line 563 is in this chain. That indicates a new direction for fixing *relevant multi-location bugs*: since modifications at each edit point possess correlation in logic and it is hard for current technologies to fix at each point, we can find out the closest intersection to the buggy points in the EPCs and utilize SMT solver to find a patch which avoids the error. It is possible to generate a patch as long as an intersection can be found no matter how many buggy points the program possesses. All we need to do is selecting out the top-k suspicious statements, calculating their EPCs, and searching for patches at the intersections preferentially. Note that two situations of *relevant multi-location bugs* have been introduced in Section II. If two AST nodes have dependency relations, one statement will appear in another's EPC; if two nodes are both under a common node, then there will be an intersection in their EPCs.

Suggestion 1: For *similar multi-location bugs*, use a suitable fitness function for guiding the repair process combined with strengthened test suite.

Suggestion 2: For *relevant multi-location bugs*, find out the intersection of several EPCs and search for modifications at that point.

B. Case Study 2: Patch of Chart#5 Generated by Nopol

The patch of Chart#5 generated by Nopol is shown in Fig. 2. Unlike the human-written patch shown in Listing 2 modifying two code chunks, this patch only modifies a conditional statement to repair this bug. The modification point is at line 563, just under the buggy point. Nopol is a semantic-based program repair tool utilizing angelic values and a Satisfiability Modulo Theory (SMT) solver for synthesizing conditional expressions. The conditional expression it generates really avoids the error. The variable *overwritten* is defined with *null* in line 546 and its value can only be modified if the condition in line 548 is met. When the condition in line 548 is not met, *overwritten* keeps the value *null* and the program goes to the conditional branch in line 563 where the condition is not satisfied, either, after being modified. Then the

V. EMERGING RESULTS

In this section, we introduce our preliminary experimental results. We first randomly selected four *similar multi-location bugs* whose test cases are not yet capable for exposing all the defects and used SimFix, the latest tool with the same fitness function as ACS contains, to perform the fixing using the strategy of Suggestion 1. We then randomly selected four *relevant multi-location bugs*, each of whose EPC is less than ten lines (due to the time limitation), added EPCs information into their bug locations as we introduced in Suggestion 2, and utilized Nopol for synthesizing patches. All the selected bugs have never been repaired and the results are shown in Table II.

We manually examine the generated patches and consider a patch correct if it is the same or semantically equivalent to human-written one. The results show that our strategies repair two bugs by applying our suggestions with two current tools.

TABLE II. PRELIMINARY RESULTS

Bug ID	SimFix + S1				Nopol + S2			
	M46	M49	L62	T3	M95	CL8	CL50	L22
Fixed?		✓				✓		

Test case purification focuses on recovering the execution of omitted assertions and thus is sometimes useless for strengthening the test suite. For example, in Math#49, the test case *OpenMapRealVector* fails for its first function invocation which leads to an *InvocationTargetException* and thus the following function invocations cannot be executed, being the reason for SimFix not fixing this bug. In our experiment, we added the test cases generated by Evosuite into test suite, successfully exposed the two defects, and at last fixed this bug. The generated patch is the same as the standard one provided by Defects4J. We performed the same operation to the other three *similar* type bugs but SimFix failed to generate patches for them for mainly two reasons. For Lang#62 and Time#3, the reason is SimFix finds for fix ingredients in the original projects but there is no similar code in the source files, indicating that we may combine source files with existing open source projects to enlarge the space for searching for fix ingredients in the future. While for Math#46, the reason is the code snippet is so large (10 lines) that it considers the donor with a return statement the same as human-written patch as not similar. However, after we adjusted the code snippet size to a finer-grained value (2 lines), it still neglected the snippet which contains fix ingredient, which indicates that the similarities for identifying donor code snippets need to be improved.

In Closure#8, the edit point in human-written patch is line 202 in class *CollapseVariableDeclarations*. We calculated the intersections of the EPCs of top-100 ranked suspicious statements and perform synthesis on these points. Finally, Nopol generated a patch which adds an if-conditional statement to avoid the error under the another class. We performed the same operations to the other three *relevant* type bugs but they failed because of not finding angelic value at the interpoints (Lang#22) and not synthesizing a patch (Math#95 and Closure#50), corresponding to two of five limitations (*No angelic value found* and *Timeout in SMT*) the authors discussed in their paper, which means the repair ability of Nopol needs to be improved.

By making some micro-adjustments of our suggestions with current tools, we fixed two multi-location bugs. The failed cases are due to the weaknesses of current tools according to our analysis, indicating the potential of our suggestions to fix more bugs when combined with more powerful tools.

VI. CONCLUSION

In this paper, we divided multi-location bugs in Defects4J into two categories according to the repair actions in their patches, summarized the situation of these bugs fixed by current tools, and learned the successful experience as well as put forward two suggestions for future research (one for each type). Guided by our suggestions, we successfully fixed two multi-location bugs in Defects4J which have never been repaired before. To our best knowledge, we are the first to propose strategies by analyzing patches generated by current tools, bringing new idea for APR techniques as well as pointing out possible ways for multi-location program repair.

ACKNOWLEDGEMENT

This work was supported by the National Natural Science Foundation of China under Grant 61672529.

REFERENCES

- [1] Weimer W, Nguyen T V, Le Goues C, et al. Automatically finding patches using genetic programming[C]//Proceedings of the 31st International Conference on Software Engineering. IEEE Computer Society, 2009: 364-374.
- [2] Qi Y, Mao X, Lei Y, et al. The strength of random search on automated program repair[C]//Proceedings of the 36th International Conference on Software Engineering. ACM, 2014: 254-265.
- [3] Le X B D, Chu D H, Lo D, et al. S3: syntax-and semantic-guided repair synthesis via programming by examples[C]//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM, 2017: 593-604.
- [4] Mehtaev S, Yi J, Roychoudhury A. Angelix: Scalable multiline program patch synthesis via symbolic analysis[C]//Proceedings of the 38th international conference on software engineering. ACM, 2016.
- [5] Sobreira V, Durieux T, Madeiral F, et al. Dissection of a bug dataset: Anatomy of 395 patches from Defects4J[C]//2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018: 130-140.
- [6] Just R, Jalali D, Ernst M D. Defects4J: A database of existing faults to enable controlled testing studies for Java programs[C]//Proceedings of the 2014 International Symposium on Software Testing and Analysis. ACM, 2014: 437-440.
- [7] Zhong H, Su Z. An empirical study on real bug fixes[C]//Proceedings of the 37th International Conference on Software Engineering-Volume 1. IEEE Press, 2015: 913-923.
- [8] Xiong Y, Wang J, Yan R, et al. Precise condition synthesis for program repair[C]//Proceedings of the 39th International Conference on Software Engineering. IEEE Press, 2017: 416-426.
- [9] Soto M, Le Goues C. Using a probabilistic model to predict bug fixes[C]//2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018: 221-231.
- [10] Jiang J, Xiong Y, Zhang H, et al. Shaping Program Repair Space with Existing Patches and Similar Code[C]// The International Symposium on Software Testing and Analysis. 2018.
- [11] Martinez M, Durieux T, Sommerard R, et al. Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset[J]. Empirical Software Engineering, 2017, 22(4): 1936-1964.
- [12] DeMarco F, Xuan J, Le Berre D, et al. Automatic repair of buggy if conditions and missing preconditions with SMT[C]//Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis. ACM, 2014: 30-39.
- [13] Xin Q, Reiss S P. Leveraging syntax-related code for automated program repair[C]//Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2017: 660-670.
- [14] Chen L, Pei Y, Furia C A. Contract-based program repair without the contracts[C]//Automated Software Engineering (ASE), 2017 32nd IEEE/ACM International Conference on. IEEE, 2017: 637-647.
- [15] Le X B D, Lo D, Goues C L. History Driven Program Repair[C]// IEEE, International Conference on Software Analysis, Evolution, and Reengineering. IEEE, 2016:213-224.
- [16] Hua J, Zhang M, Wang K, et al. Towards practical program repair with on-demand candidate generation[C]//Proceedings of the 40th International Conference on Software Engineering. ACM, 2018: 12-23.
- [17] Xuan J, Monperrus M. Test case purification for improving fault localization[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014:52-63.
- [18] Galeotti J P, Fraser G, Arcuri A. Improving search-based test suite generation with dynamic symbolic execution[C]//Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on. IEEE, 2013: 360-369.
- [19] Castro M, Liskov B. Practical Byzantine fault tolerance[C]// Symposium on Operating Systems Design & Implementation. ACM, 1999:173-186.
- [20] Guo A, Mao X, et al. An Empirical Study on the Effect of Dynamic Slicing on Automated Program Repair Efficiency[C]// In: Proceedings of IEEE International Conference on Software Maintenance and Evolution. IEEE, 2018: 580-584.

Logical Segmentation of Source Code

Jacob Dormuth, Ben Gelman, Jessica Moore, David Slater

Machine Learning Group

Two Six Labs

Arlington, Virginia, United States

E-mail: {jacob.dormuth, ben.gelman, jessica.moore, david.slater}@twosixlabs.com

Abstract

Many software analysis methods have come to rely on machine learning approaches. Code segmentation - the process of decomposing source code into meaningful blocks - can augment these methods by featurizing code, reducing noise, and limiting the problem space. Traditionally, code segmentation has been done using syntactic cues; current approaches do not intentionally capture logical content. We develop a novel deep learning approach to generate logical code segments regardless of the language or syntactic correctness of the code. Due to the lack of logically segmented source code, we introduce a unique data set construction technique to approximate ground truth for logically segmented code. Logical code segmentation can improve tasks such as automatically commenting code, detecting software vulnerabilities, repairing bugs, labeling code functionality, and synthesizing new code.

1 INTRODUCTION

With the proliferation of open-source development practices and code sharing services, such as GitHub and Bitbucket, large bodies of source code are increasingly available to developers. There are a number of ways in which these code corpora could assist with the software development process; of particular interest is the application of machine learning to software engineering practices. Recent literature has utilized machine learning with source code at scale, developing tools to generate comments [7], detect software vulnerabilities [11], repair bugs [4], label functionality [2], and synthesize new code [6]. Code segmentation - the process of decomposing source code into meaningful blocks - can augment these methods, for instance, by determining what portions of a file are functionally similar,

identifying where to generate automatic comments, and locating useful sub-function boundaries for bug detection.

Current approaches to segmentation do not intentionally capture logical content that could improve its usefulness to the aforementioned problems. Traditionally, code segmentation has been done at a syntactic level. This means that language-specific syntax, such as the closing curly brace of a class or function, is the indicator for a segment. Although this method is conceptually simple, the resulting segments do not intentionally take into account semantic information. Syntactic structures in natural language, such as sentences and paragraphs, are generally also indicators of semantic separation. In other words, two different paragraphs likely capture two logically separate ideas, and it is simple to delineate them by splitting at the end of each paragraph. Locating logical segments in source code, however, is a non-trivial task. Syntactic structures, such as a for-loop followed by an if-statement, are not particularly indicative of semantic changes in the code. Determining if the for-loop and if-statement are working to achieve the same logical task is a more difficult problem than the paragraph delineation analogue. Logical code segmentation captures arbitrary combinations of syntactic structures in a single segment.

Logical code segmentation has a multitude of uses, including: improving code search tools by returning relevant segments of code instead of entire files or functions, recommending locations to add comments, classifying the functionality of source code by reducing the problem space from entire files/projects to concentrated blocks of code, or using the segments as features for a model that attempts to determine the modularity or complexity of a given code file. Logical segments are able to featurize code, reduce noise, and limit a problem space to certain types of segments.

In this work, we develop a novel code segmentation method to generate logical segments in a language-agnostic fashion. Although language specificity may help model code by allowing options such as language-specific tokens and abstract syntax trees, it adds a non-trivial burden to operationalizing tools. Language specific approaches require

training many models, each with their own parsers, training sets, and hyperparameters. Our language-agnostic approach is able to avoid those extra steps, improving the generalizability, availability, and ease-of-use of the results. Because there is no data set of source code that is conveniently split into logical segments, we first establish a unique data set construction process using Stack Overflow¹ (SO). Human curation is critical to determining logical segments, and Stack Overflow provides that at a vast scale.

We build on prior work in natural language processing, training a bidirectional long short-term memory (LSTM) neural network to split source code into logical segments. Since this is the first work to perform this segmentation in the source code domain, we provide baseline models and multiple deep neural networks for comparison. We validate the results on six programming languages to display the language agnosticism of the method. Lastly, we qualitatively discuss the model’s results on real source code documents to provide insight on performance in the desired domain.

Our main contributions are as follows:

- A novel data set construction method that utilizes crowd-sourced data to approximate logical segments.
- First work, to our knowledge, to perform logical code segmentation regardless of syntactical correctness or programming language.
- Baseline models for the logical segmentation problem, demonstrating the relative effectiveness of our language-agnostic, deep neural network on six different programming languages.

2 RELATED WORK

The literature on source code segmentation is relatively sparse, with existing work offering syntax-specific solutions and requiring language specificity. Ning, Engberts, and Kozaczynski develop a code browser that allows users to select useful, reusable segments by analyzing the control flow and data flow in the abstract syntax trees (ASTs) of COBOL programs [9]. More recently, Wang, Pollock, and Vijay-Shanker attempt to locate logical segments by developing rules that look for specific syntactical patterns [12]. Wang et al. generate an AST for a given Java method, apply rules that analyze the data flow and syntactic structures, and then add line breaks around the resulting segments to enhance the readability of the method. Although these methods provide sophisticated analysis and rules, they are inextricably linked to the language that they operate on, and thus will not transfer to other languages.

In this work, we deviate from the practice of using ASTs as the foundation for our analysis. By treating source code

as a body of text akin to an NLP problem, we avoid any programming-language-specific challenges posed by other methods. Text segmentation has been researched more thoroughly than the source code analogue, with methods ranging from LDA [10], to semantic relatedness graphs [3], to deep learning approaches [1]. Of particular note is the use of bidirectional LSTMs to identify the breaks between segments of Wikipedia articles [8]. An LSTM is a recurrent neural network that processes sequential information [5]. Although we use the bidirectional LSTM as the primary framework of our model, we make several changes to adapt to the source code domain. We use a character embedding instead of the commonly used word embedding: due to the vast number of possible unique identifiers in source code, optimizing an embedding for each token is infeasible. Additionally, we require an evaluation metric specifically based on our data generation method to represent how well the model recognizes segments the method creates.

3 DATA SET GENERATION

We model code segmentation as a classification problem, where given a sequence of characters, we must predict whether a character denotes the beginning of a new code segment. In order to generate training data suited to the task, we use Stack Overflow, a forum where users can ask questions, receive answers, and post code snippets on a wide range of computer programming topics. Because the posters are focused on answering a specific question, the code snippets are generally geared towards a single logical task. Fig. 1 shows an example response on Stack Overflow containing blocks the user has marked as code. We pull code snippets by searching for posts tagged with six different programming languages: C, C++, Java, Python, Javascript, and C#.

One problem with the data, however, is that the distribution of the number of lines per code snippet is heavily skewed. Fig. 2 shows that the majority of code snippets are only a few lines long. Using the entire data set could bias the model to predict segments every few lines because the model will have seen so many short code snippets, which is unlikely to be the case in real source code files. Using only very long snippets, however, would not leave much training data. Thus, we heuristically filter out all code snippets that are less than four lines long using the elbow method.

After filtering, we generate segments by concatenating snippets with a newline character, thereby marking the beginning of a new segment. We refer to these as “dividing newlines.” It is important to note that not all newlines mark a new segment because a single code snippet may contain many newlines. After this process is complete, the result is essentially a giant block of concatenated code snippets. To obtain individual data points, we iterate through the block

¹<https://stackoverflow.com>

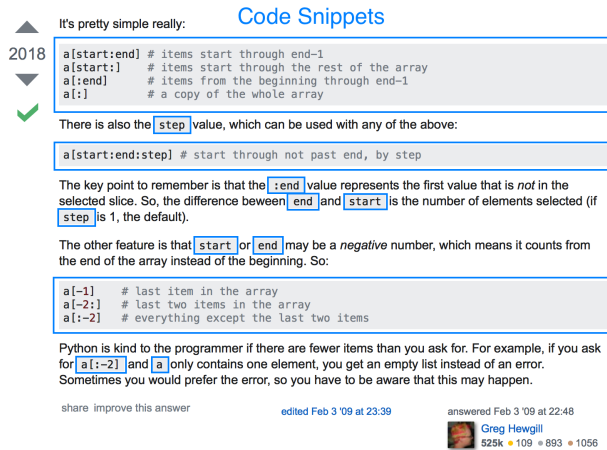


Figure 1. Example Stack Overflow post showing code snippets.

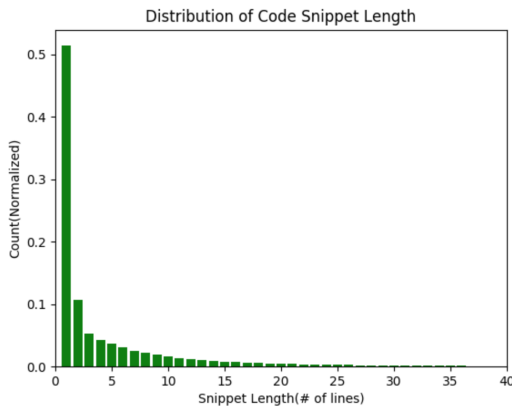


Figure 2. Graph depicting the heavily skewed snippet length distribution on Stack Overflow.

of snippets and generate data points using three methods: bag of characters, uncentered, and centered, shown in Fig. 3.

In the bag of characters method, one data point is created by taking 7 lines from the block of snippets and counting the characters to create a “bag of characters” for each line. These bags of characters are then concatenated together to create a single training sample. If the middle newline (the fourth of seven) is a dividing newline, then the label for this data point is a 1. Otherwise, the label is a 0. This process is repeated by sliding the 7 line window forward by 1 line. We track the counts of 256 unique characters, corresponding to all the ASCII characters, with an average of about 29 characters per line.

In the uncentered method, we create a data point by selecting all the characters in a 100-character window. In this

process, we assign one label for each character in the window, meaning one data point has 100 characters and 100 labels. If the character is a dividing newline, the corresponding label is a 1. This process is repeated by sliding the window by 100 characters. It is important to note that there is no guarantee a data point in this method will contain a dividing newline (or any newline at all). Even when the data point does contain a dividing newline, there is no guarantee that it will occur in the center of the input.

In the centered method, we create a data point by locating a newline and taking a window of 50 characters before and after that newline, for a total of 101 characters. If that newline is a dividing newline, then the label for this data point is a 1. Otherwise, the label is a 0. This process is repeated by centering the window on the next occurring newline. Whilst iterating through the block of snippets, any window that does not have a newline in the center will be ignored. It is important to note that other newlines, including dividing newlines, may occur in the window; however, the label is assigned based only on the center newline.

4 METHODOLOGY

We experiment with three models: a logistic regression model (to serve as a baseline), and two neural network architectures that utilize bidirectional LSTM layers. Every model is trained on seven training sets. Six of the training sets correspond to the different languages: C, C++, Java, Python, Javascript, and C#. The last training set combines all of the languages in order to evaluate language agnosticism.

The logistic regression model utilizes the bag of characters data format for training. Each bag of characters, corresponding to one line of code, contains counts for 256 unique characters. The model takes seven bags of characters, for a total input size of 1,792. The output is a value from 0 to 1, representing the probability that the bag of characters corresponding to the fourth line (the middle line out of 7) contains a dividing newline (recall the labeling scheme from section 3).

The first neural network architecture uses the uncentered data format for training. This model has an input layer of length 100, one for each character in an uncentered data point. Each character is passed through a 20-dimensional character embedding, which converts a character to a 20-dimensional, real-valued vector. These embeddings are passed to a bidirectional LSTM layer of size 256. A bidirectional LSTM allows for past and future information to be used together, whereas a standard LSTM only considers past information. If a human were to segment source code, they would likely look ahead and use future information to piece together their decisions. The sequential information that the bidirectional LSTM learns is condensed using three

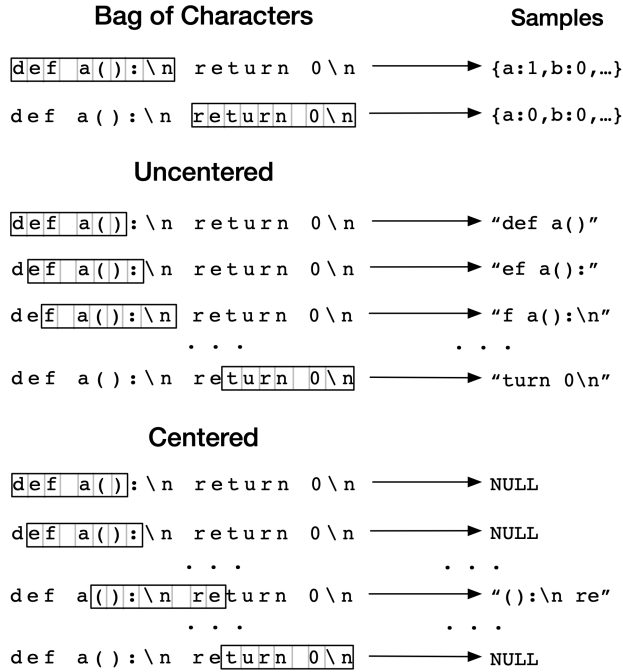


Figure 3. The three data generation methods operating on the same piece of code. The window sizes are shortened for visual clarity. In the bag of characters method, each sample is the concatenation of seven bags of characters corresponding to seven consecutive lines. In the uncentered method, each sample is simply all the characters in the window. In the centered method, each sample is all the characters in a window where the middle character of the window is a newline. If a window in the centered method does not contain a newline in the center, no sample (“NULL”) is generated.

time-distributed dense layers, sizes 150, 75, and 1, respectively. Since the layers are time distributed, the last layer has 100 total outputs (one for each time step). Each output is the probability that the character at that time step indicates the beginning of a new logical segment. The uncentered data format has 1 label for each character, so the 100 outputs and 100 labels are used to compute the loss. We use a batch size of 128, a dropout strength of 0.2 between each layer for regularization, and binary cross entropy as our loss function.

The second neural network architecture uses the centered data format for training. In terms of model structure, it is nearly identical to the uncentered model except that the input size is 101 characters and that it has one additional layer. The last layer of the centered model is a dense layer that maps the 100 time distributed outputs (last layer of the uncentered model) to a single output. Because the centered data format guarantees a newline at the center of the input sequence, only a single output is required from the centered

model. That single output and the label for the center newline are used to compute the loss.

4.1 Evaluation

Due to our data generation method, only a newline character can denote the beginning of a new code segment. As a result, we specifically measure the newline accuracy of our models. This distinction is critical because the logistic regression and centered models predict only on newlines, while the uncentered model outputs a prediction for every character, which would skew accuracy. The newline accuracy metric is the percentage of newlines that are classified correctly as either a dividing newline or non-dividing newline.

We split the collection of code snippets into train/validation/test sets of 80% / 10% / 10%. The training process is stopped when the model does not improve after 20 epochs.

5 RESULTS

In total, we train 21 different models: 18 single-language models and 3 multi-language models. For each of the three architectures (logistic regression, uncentered LSTM, centered LSTM), we train a model on each language individually, as well as a model on all the languages simultaneously. Table 1 displays the newline accuracies for each model/language pairing.

The logistic regression models perform the worst, but are still significantly better than predicting a non-dividing newline every time (non-dividing newlines are the most common). This is an expected result because the logistic regression models do not take into account character interactions like the more complex models. The neural network models show a significant improvement in performance over the logistic regression baselines.

Although the single-language models generally perform slightly better than their multi-language counterparts, the difference in newline accuracy is relatively small. This is a very positive result because the multi-language models are able to discern logical segments across languages with little to no performance hit.

Another interesting comparison is the difference in performance between the uncentered and centered models. In the single-language category, the uncentered models outperform the centered models across the board. On the other hand, in the multi-language category, the centered models are usually more effective. One possible reason for this is the difference in code context between uncentered and centered data points. In the uncentered data format, it is possible for a data point to contain no newlines whatsoever. This may help the model understand the difference

Table 1. Newline Accuracies of Each Model

Model	Language					
	<i>C</i>	<i>C++</i>	<i>Java</i>	<i>Python</i>	<i>Javascript</i>	<i>C#</i>
Single-Language Logistic Regression	95.26	95.2	95.6	91.61	94.53	95.63
Single-Language Uncentered LSTM	98.8	98.76	98.96	99.17	99.2	99.3
Single-Language Centered LSTM	97.35	98.35	98.84	97.82	98.79	98.87
Multi-Language Logistic Regression	94.76	94.81	95.1	90.76	93.48	95.09
Multi-Language Uncentered LSTM	98.33	98.56	98.75	97.98	98.65	98.86
Multi-Language Centered LSTM	98.5	98.62	98.82	97.83	98.73	98.95
Percent Non-Dividing Newlines	92.06	92.12	92.35	91.10	92.02	92.12

Table 1. Newline accuracy test results for every model and language. “Single-Language” models are trained and tested on one language at a time. “Multi-Language” models are trained on every language simultaneously and then tested on each language separately. “Percent Non-Dividing Newlines” is the newline accuracy if a model were to always predict non-dividing for every newline.

in context when a newline is actually present. In the multi-language scenario, however, snippets with no newlines may have a fundamentally different pattern for each language in the data set, which may significantly complicate what the model needs to learn. The centered multi-language models only need to learn the differences between languages as it pertains to the context around a newline, reducing the cost of learning on multi-language data. Given enough data and time, it may be possible that the multi-language models would be able to utilize the uncentered snippets more effectively.

One other noteworthy observation is the impact of similarity across languages. In the multi-language scenario, Python is the most syntactically distinct language and consistently performs the worst. It is feasible that the models are able to transfer knowledge across languages, so similar languages may benefit from each other’s data. The differences in results between languages may also speak to the quality of Stack Overflow snippets for those languages. It is possible that different programming languages attract different questions, topics, and code snippet qualities on Stack Overflow, which would ultimately influence the model’s performance.

5.1 Testing on Source Code

In order to better understand the utility of these models, Fig. 4 and 5 showcase examples of the single-language deep learning model running on a Python source file.

In Fig. 4, the model is able to recognize when the functionality of the code changes from defining a Keras model to loading the weights and compiling the model. Fig. 5 shows how the model is able to differentiate between a string operation in a for-loop and a new task of opening and writing

```
bimodel.add(TimeDistributed(Dense(75, activation = 'relu'))
bimodel.add(Dropout(.2))
bimodel.add(TimeDistributed(Dense(1, activation = 'sigmoid'))

filename = "/home/jdormuth/reversecrowd/trials/small-snippets/weights-
improvement-09-0.0030.hdf5"
bimodel.load_weights(filename)
bimodel.compile(loss = 'binary_crossentropy', optimizer='adam')
```

Figure 4. The model recognizes that compiling and loading the weights of a Keras model is a different task from defining the layers of the model. The dashed line is the model’s prediction of the segment location.

```
for x in range(len(divisions)-1,-1,-1):
    raw_code = replace_str_index(raw_code,divisions[x])
-----
text_file = open("source-code-test.txt", "w")
text_file.write(raw_code)
text_file.close()
```

Figure 5. The model distinguishes between string operations and writing to a file. The dashed line is the model’s prediction of the segment location.

to a text file.

6 CHALLENGES AND LIMITATIONS

One of the biggest challenges is the lack of ground truth logical segments. Because the Stack Overflow code snippets are simply concatenated together, the resulting data is not necessarily representative of real code. It is possible that two snippets concatenated together do not result in syntactically correct code. There is also no guarantee of standard formatting practices that one would expect to see in formal software projects.

Additionally, the data generation method assumes that a

code snippet represents a logical chunk of code. Although this is usually the case, there is no way to guarantee that all code snippets are segmented logically. It is possible that an individual snippet contains multiple logical tasks. It is also possible that two randomly selected snippets perform similar tasks; this introduces noise to the data set because they will still be labeled with a dividing newline.

One possible way to address these challenges is to use in-line comments in source code files as logical division points. There is intended meaning behind the placement of comments in source code files, whereas the concatenation of Stack Overflow code snippets is randomized. This method could better reflect the properties of source code.

7 CONCLUSIONS AND FUTURE WORK

We present a novel method to perform logical segmentation of source code. Using crowd-sourced data from Stack Overflow, we create a unique data set construction technique to approximate logical segments in source code. Drawing from the NLP domain, we develop deep neural network models utilizing bidirectional LSTMs that can predict on source code regardless of language or syntactical correctness. Lastly, we provide baselines and an appropriate metric to evaluate the performance of our models with regard to our data set construction.

Although our method is the first success in language-agnostic logical code segmentation, there are a variety of potential architectural and parameter improvements. We could incorporate an attention mechanism into the LSTM, allowing the model to learn more specific features in the source code. Another avenue could be adjusting model parameters such as window size, network depth, and loss functions. For example, the loss function of the segmentation models could be modified to incorporate the error term of another task that uses segments as input. Finally, input representations that are larger than character-scale may improve the models without significant increases in model complexity; word or token embeddings in addition to the character embedding may be able to achieve this.

ACKNOWLEDGEMENTS

This project was sponsored by the Air Force Research Laboratory (AFRL) as part of the DARPA MUSE program. We would like to thank Robert Gove and Casey Haber for their valuable feedback, support, and contributions to figures. We also thank Banjo Obayomi for infrastructure support.

References

- [1] P. Badjatiya, L. J. Kurisinkel, M. Gupta, and V. Varma. Attention-based neural text segmentation. In *Eu-*

ropean Conference on Information Retrieval, pages 180–193. Springer, 2018.

- [2] B. Gelman, B. Hoyle, J. Moore, J. Saxe, and D. Slater. A language-agnostic model for semantic source code labeling. In *Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis*, pages 36–44. ACM, 2018.
- [3] G. Glavaš, F. Nanni, and S. P. Ponzetto. Unsupervised text segmentation using semantic relatedness graphs. Association for Computational Linguistics, 2016.
- [4] J. Harer, O. Ozdemir, T. Lazovich, C. Reale, R. Russell, L. Kim, et al. Learning to repair software vulnerabilities with generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 7944–7954, 2018.
- [5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [6] V. Kashyap, R. Swords, E. Schulte, and D. Mel-ski. Musynth: Program synthesis via code reuse and code manipulation. In *International Symposium on Search Based Software Engineering*, pages 117–123. Springer, 2017.
- [7] J. Moore, B. Gelman, and D. Slater. A convolutional neural network for language-agnostic source codesummarization. In *ENASE (to appear)*, 2019.
- [8] N. Mor, O. Koshorek, A. Cohen, and M. Rotman. Learning text segmentation using deep lstm. 2017.
- [9] J. Q. Ning, A. Engberts, and W. V. Kozaczynski. Automated support for legacy code understanding. *Communications of the ACM*, 37(5):50–58, 1994.
- [10] M. Riedl and C. Biemann. Text segmentation with topic models. *Journal for Language Technology and Computational Linguistics*, 27(1):47–69, 2012.
- [11] R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood, and M. McConley. Automated vulnerability detection in source code using deep representation learning. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 757–762. IEEE, 2018.
- [12] X. Wang, L. Pollock, and K. Vijay-Shanker. Automatic segmentation of method code into meaningful blocks to improve readability. In *2011 18th Working Conference on Reverse Engineering*, pages 35–44. IEEE, 2011.

TL-GAN: Generative Adversarial Networks with Transfer Learning for Mode Collapse

1st Xianyu Wu

College of Computer Science
Chengdu University of Information Technology
Chengdu, China
Email: xianyuWU42@163.com

2nd Shihao Feng

College of Computer Science College of Information Science and Technology
Southwest University
Chongqing, China
15340590451@163.com

3rd Canghong Shi

College of Information Science and Technology
Southwest Jiaotong University
Chengdu, China
canghongshi@163.com

4rd Xiaojie Li*

College of Computer Science
Chengdu University of Information Technology
Chengdu, China
lixj@cuit.edu.cn

5th Jing Yin

College of Computer Science
Chongqing University of Technology
Chongqing, China
yinjing@cput.edu.cn

6th Jiancheng Lv

College of Computer Science
Sichuan University
Chengdu, China
lvjiancheng@scu.edu.cn

Abstract—Image generation based on the generative adversarial network (GAN) has been widely used in the field of computer vision. It helps generate images similar to the given data by learning their distribution. However, in many tasks, training on small datasets of scenes may lead to mode collapse, such that the generated images are often blurred and almost the same. To solve this problem, we propose a generative adversarial network with transfer learning for mode collapse called TL-GAN. Owing to the size of the training dataset, we introduce transfer learning (VGG pre-training network) to extract more useful features from the underlying pixels and add them to the discriminator, which can be used to calculate the distance between samples, and to provide the discriminator with a new training target. The discriminator thus learns the best features that can distinguish between real data and generated data using the proposed model. This also enhances the learning capability of the generator, which learn further about the distribution of real data. Meanwhile, generator can produce new images more realistic. The results of experiments show that the TL-GAN can guarantee the diversity of samples. A qualitative comparison with several prevalent methods confirmed its effectiveness.

Index Terms—Image generation, Generative Adversarial Network, Mode collapse, Transfer learning, VGG pre-training network

I. INTRODUCTION

Most advanced deep neural network algorithms can learn highly complex problems and patterns [1]–[5], and their capabilities are impressive. However, humans can do far more than these algorithms on image recognition and speech recognition tasks, and it had appeared unlikely that such task could be automated. However, GANs [6]–[8] have made this possible.

Generative Adversarial Network (GAN) is an unsupervised method [9] that is among the most successful computer vision

algorithms. It has been extensively researched and developed, especially in the context of image generation [10]–[13]. GAN was proposed by Ian Goodfellow in 2014, and consists of two parts: a generator network G and a discriminator network D . The generator network generates samples by learning the distribution of real data. Its training objectives include maximizing the probability of error. The goal of the generator is to deceive the discriminator by generating images from random noise that are similar to real data, and the discriminator attempts to learn the differences between the generated samples and real ones.

Many applications of the GAN have been proposed [8], [14], [15]. For example, the iGAN contains two kinds of guidance information [8]: to paste the high-definition texture in the original image on the shape of the given object in it by using light field information in a different space. The texture on the paste is iterated until the resulting image appears realistic. Like the iGAN, the GP-GAN copies and pastes images directly [14], better integrates them into the original images and performs blending. However, it is a supervised training model. In the process of learning to blend, there is a supervised goal and a supervised loss function. CycleGAN allows two domains to transform each others images [15]. Traditional GAN features one-way generation while CycleGAN involves mutual generation. It is called a cycle because its network is ring-like. The input to CycleGAN is a pair of images.

Despite its wide application, it has many problems, such as training instability, gradient disappearance, mode collapse and so on. It is an important failure mode. Specifically, model collapse is called the most critical failure mode in GAN networks. Generators may reproduce exactly the same image, called pattern crash. Generally speaking, real world data distribution is highly complex and multimodal. The probability distribution described by the data has several "peaks" with different sub-groups of samples. The generator folds into a very narrow distribution, and the generated samples are no

This study was supported by the National Natural Science Foundation of China (Grant No. 61602066) and the Scientific Research Foundation (KYTZ201608) of CUIT, and in part by the major project of the Education Department in Sichuan (17ZA0063 and 2017JQ0030), and the Sichuan International Science and Technology Cooperation and Exchange Research Program (2016HH0018). DOI reference number: 10.18293/SEKE2019-160

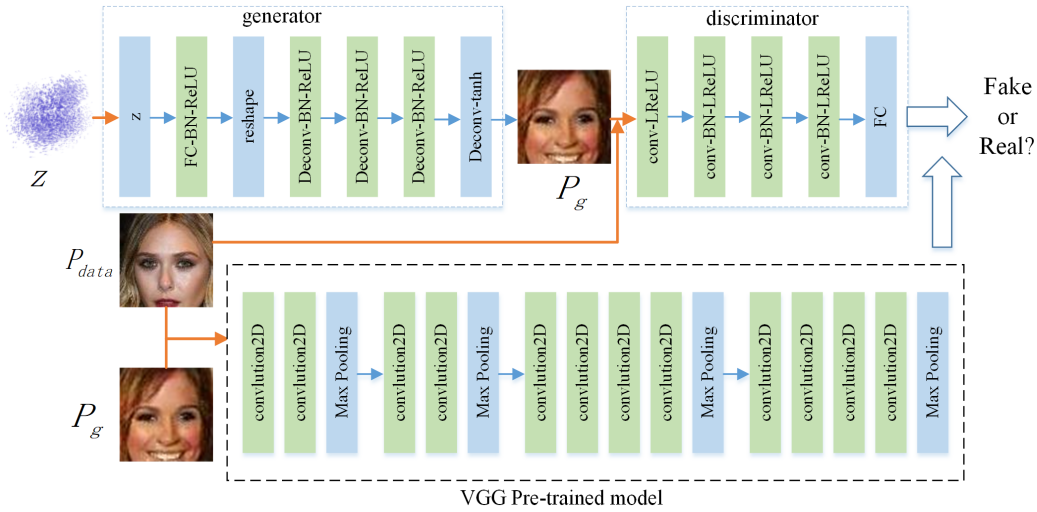


Fig. 1. Model architecture. z denotes random variables from a uniform or a Gaussian distribution, and P_g denotes fake images generated by the generator network. P_{data} represents real data, and conv and deconv denote the convolutional and the deconvolutional layers, respectively. FC denotes a fully connected layer.

longer changed. This obviously violates GAN's essence. For example, for face generation, we want to train a GAN that can generate face images. Generators need to learn hundreds of thousands or even millions of faces. However, if there are only tens of thousands of training data sets, the generator may collapse, resulting in poor diversity in the generated samples, which limits the usefulness of learning in GAN.

To solve the problem of mode collapse on small datasets in the GAN, we propose in this paper generative adversarial networks with transfer learning for mode collapse (TL-GAN). We introduce transfer learning (VGG pre-trained model) to extract useful features from the underlying pixels in images and add them to the discriminator. These features can be used to calculate the distance between samples and provide the discriminator with a new training target. This also enhances the learning capability of the generator, which can then learn more about the distribution of real data. Instead of encouraging samples generated by each generator to approach a single maximum likelihood, generator collapse is avoided. The overall performance of these samples are closer to the real image, and a suitable distance between different samples in the space can be ensured.

II. RELATED WORK

Although it is widely used, there are problems with the GAN, such as difficulty in training, mode collapse, and a lack of diversity in the generated samples. Several variants of the GAN have been proposed to solve these problems [10]–[13].

DCGAN: Since it was proposed in 2015, the DCGAN has been widely used in many applications, and significantly improves the stability of the training of the GAN and the quality of its results [10]. However, this only solves the problem temporarily and does not resolve difficulties in train-

ing. Furthermore, it is not easy to train D and G to reach equilibrium.

LSGAN: Proposed in 2016, the LSGAN uses the least-squares loss function instead of the loss function of the GAN to alleviate instability in its training and the lack of diversity that leads to poor image quality [11]. However, the LSGAN also has drawbacks. Its excessive penalty for outliers may lead to a reduction in the diversity of sample generation. It is likely to yield a simple imitation with minor modifications to the real data.

WGAN: The WGAN, developed in 2017, improves GAN in term of the loss function [12]. It removes sigmoid from the last layer of the discriminator, and forcibly truncates the updated weights to a certain range to meet the conditions of Lipschitz continuity. However, clipping the weight parameters blindly to guarantee stable training can lead to low-quality and low-resolution images.

WGAN-GP: The WGAN-GP, developed in 2017, is an improvement over the WGAN [13], specifically its continuity constraints. The contribution of WGAN-GP is that a technique to restrict Lipschitz continuity-the gradient penalty is proposed to solve the problem of mode collapse with a vanishing training gradient. While the WGAN-GP is an improvement over WGAN in some cases, it is not significantly superior to reliable GAN methods in terms of results. A number of researchers have attempted to solve problems in the GAN with mixed success.

III. GAN WITH TRANSFER LEARNING

A. Approach

In general, the GAN model uses a large number of training datasets, of the order of hundreds of thousands of images (CelebA, LSUN) [11], [16]. In such cases, the generated samples are diverse and high quality and conform to the

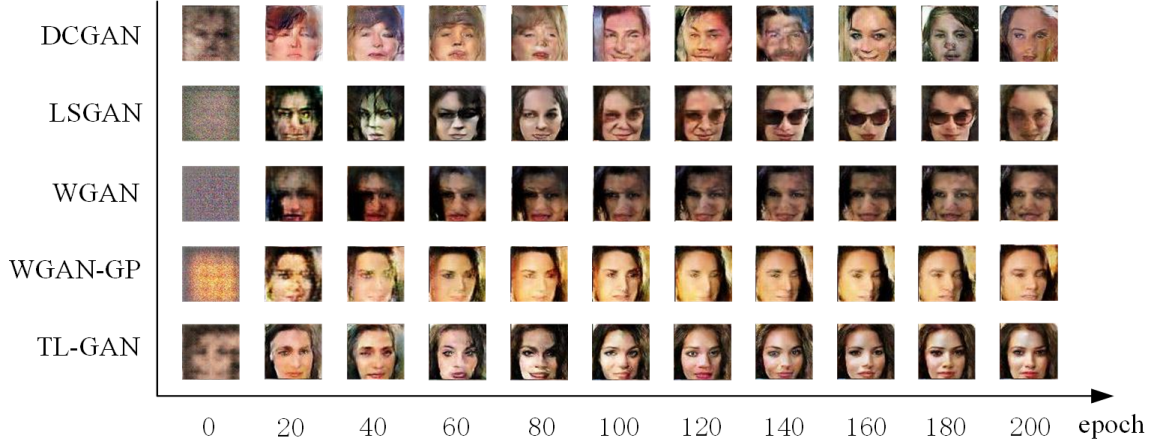


Fig. 2. Comparison of the DCGAN, LSGAN, WGAN, and WGAN-GP with our method for different epochs of a single sample on the CelebA dataset.

distribution of real data. However, in the case of a small training dataset, the generator network can learn only a few useful features such that the generated images look very fuzzy or even lead to mode collapse. To solve this problem, we introduce transfer learning that can transfer the influence of our knowledge on the visual perception of image generation tasks using a transmission parameter [17], [18]. The GAN with a model pre-trained on millions of image datasets using ImageNet outperforms a zero-trained depth model on the same small dataset. We use the trained VGG pre-trained model on real and generated images.

B. Model Architecture

Our network structure is shown in Figure 1. It comprises the generator network G , discriminator network D , and a VGG pre-trained model. The target of G captures the potential distribution of real data samples I_{data} , in addition to generating new data samples I_g . It starts with a random vector z as input to the generator network followed by a fully connected layer and subsequently employs a high-dimensional tensor with 512 feature maps. Note that we use the batch normalization layer for each deconvolutional layer, and the activation function makes use of the rectifier linear unit (ReLU) except in the last deconvolution layer. D is a dual classifier and determines whether the input to it is from the real data I_{data} or generated samples I_g . The discriminator network also makes use of multiple convolutions and uses the LeakyReLU as an activation function. At the end of the convolution, the tensor is stretched into a vector, which uses a fully connected layer. Our model employs a kernel of size 5×5 in both the generator and the discriminator. The VGG pre-trained model contains stacked convolution and coupled pooling layers. P_{data} and P_g are used as inputs to the VGG pre-trained model based on transfer learning to extract more high-level features. In the training process, the discriminator network aims to identify real and generated images, while the generator network attempts to make the generated samples more real with the aim of achieving the Nash equilibrium.

C. Loss function

1) *Adversarial loss*: Adversarial loss can maximize the probability of successfully determining whether a given image is from the training data or the generated samples. The goal of D is to separate images generated by G from real images. In this way, G and D constitute a min-max game. To better deceive the discriminator network such that it generates perceptually realistic images, we formally denote by D the adversarial loss of the discriminator network and by G that of the generator network. We define the adversarial losses of \mathcal{L}_d and \mathcal{L}_g as follows, respectively:

$$\mathcal{L}_d = E_{I_{data} \sim p_{data}(I_{data})} [D(I_{data})] + E_{I_g \sim p_g(I_g)} [1 - D(I_g)], \quad (1)$$

$$\mathcal{L}_g = E_{I_g \sim p_g(I_g)} [1 - D(I_g)], \quad (2)$$

where \mathcal{L}_g is the total loss of G , $D(\cdot)$ represents the output of the discriminator network, and $G(\cdot)$ is generated by G , which needs to learn real data distribution and produce more realistic images. When D cannot distinguish between real data and generated samples, the generator can better deceive the discriminator network. The final optimization of D is given by Eq. (3):

$$D^* = \mathcal{L}_d + \lambda \mathcal{L}_p, \quad (3)$$

where D^* is the total loss of D , and λ represents a constant.

2) *Perceptual loss*: We propose perceptual loss based on the VGG pre-training network to extract useful features from the underlying pixels and add some of them to the discriminator. These features are used to calculate the distance between samples and give the discriminator a new training target. It can thus solve the problem of blurred images and even mode collapse on small datasets. Moreover, the computational overhead of perception loss can be reduced by reusing the features

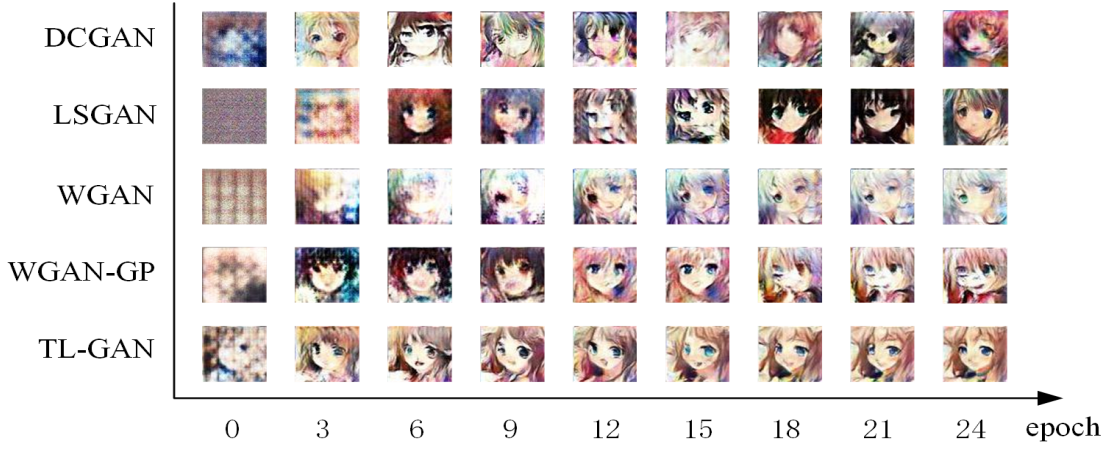


Fig. 3. Comparison of DCGAN, LSGAN, WGAN, and WGAN-GP, with our method for different epochs of a single sample on the Cartoon dataset.

extracted by the discriminator. To introduce the formula for perceptual loss, we define perceptual loss as:

$$\mathcal{L}_p = |V(I_{data}) - V(I_g)|, \quad (4)$$

where \mathcal{L}_p represents the perceptual loss, and $V(\cdot)$ represents the output of the VGG19 network.

IV. EXPERIMENTS

A. Details of Training

We used the CelebA and Cartoon datasets for training. In each training batch, we randomly selected 64 image patches as I_{data} patches, where each patch had a size of 64×64 . The random vector satisfied a uniform or a Gaussian distribution $P_z(z)$ as input to the generator network. To stabilize the network, the range of intensity of the input and output images was set to $[-1, 1]$.

TABLE I
STATISTICS OF THE DATASETS.

Datasets	Samples	Size
CelebA	10,590	64×64
Cartoon	51,223	64×64

The CelebA face dataset is an open dataset of the Chinese University of Hong Kong [16]. It contains 202,599 face images with 10,177 celebrity identities. For our experiment, we selected only part of the data as a small dataset for training, with 10,590 images. The Cartoon face dataset was downloaded from Konachan.net, a well-known animation gallery, with a total of 51,223 images. Our experiments directly used the Cartoon face dataset shared by HE [19].

Our experiments were implemented in Python on the TensorFlow framework using an NVIDIA Tesla M40 GPU to accelerate training, using an SGD at a learning rate of 0.001 for all layers to analyze model performance. The momentum parameter was set to 0.3, weight decay to 0.0001, and 64 was set as the cardinality of each patch of training. All weights

were truncated for a positive distribution and the standard deviation was 0.02. All convolutional layers were followed by leaky rectified linear units (LReLU) with a slope of 0.2. Training the CelebA and the Cartoon datasets took 4 and 3 hours, respectively.

B. Comparisons and Analysis

To verify the effectiveness of the proposed model, we compared it with state-of-the-art methods, including the DCGAN [10], LSGAN [11], WGAN [12], and WGAN-GP [13]. It is worth pointing out that there are no quantitative indicators to refer to in previous papers because the generated synthetic data have no corresponding real data reference. There was no clear measure to assess the results of image generation tasks. Inspired by [10]–[13], we used the final generated images for comparison. Our experiments contained two parts. In the first, we analyzed the effectiveness of our method on two small datasets. We analyzed changes in a single sample in the training process, and we studied the final visual effects. In the second part, we considered the stability of training in several methods.

1) *Evaluation of Visual Results: CelebA* We trained our model for 200 epochs on the CelebA dataset. For a fair comparison, we list the generated samples for the same steps in each epoch using several methods (see Figure 2). In the final epoch, we observed that the DCGAN [10], LSGAN [11], WGAN [12], and WGAN-GP [13] produced poor results on a single sample. Because the dataset was small and training time limited, the DCGAN experienced mode collapse. It generated images of faces of females and males alternately. The training process of the LSGAN was very unstable. It first generated the images of a face not wearing glasses, then one with glasses, and again one without glasses. The WGAN and WGAN-GP generated blurry images of faces because of slow convergence. On the contrary, TL-GAN generated clear images of faces from random noise in a single sample.

Cartoon We trained our model for 24 epochs on Cartoon dataset. We list the generated samples in the same steps in

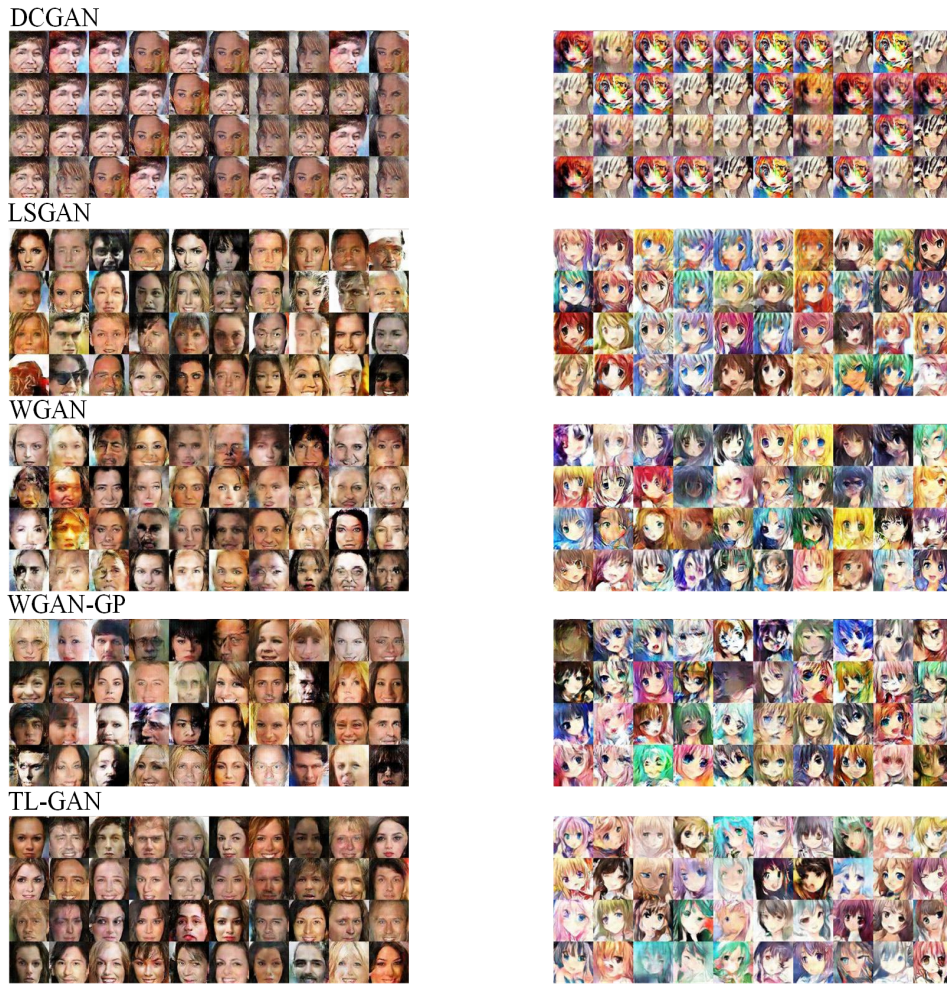


Fig. 4. Visual comparison of all samples on training dataset: CelebA dataset on the left and Cartoon dataset on the right.

each epoch from several methods. From Figure 3 it is clear that the samples generated by our method showed a steady increment in quality, and produced a perceptually realistic image. However, the phenomenon of mode collapse appeared in some other methods (DCGAN and LSGAN). We also found that all other methods except the WGAN were unstable, their generated samples did not follow a fixed trend in the generation of shapes of face images, and they constantly changed in the training process. The DCGAN and LSGAN yielded images of faces of different cartoon characters in the training process, and two types of such images were generated in training by the WGAN-GP.

Samples of the CelebA and Cartoon datasets that were used are shown in Figure 4. The DCGAN on CelebA exhibited mode collapse, and it lacked sample diversity, and thus many generated samples were almost identical. LSGAN experienced a moderate mode collapse, and some of the images of faces generated by it look very odd. The WGAN trained very slowly on the CelebA dataset and its resulting images were very blurred. A slight mode collapse was observed in the WGAN on the Cartoon dataset. For example, the images of some faces

were unusual and hairy body parts had unnatural details. Its results were better than those of the DCGAN and LSGAN, but the overall effect was unsatisfactory. The WGAN-GP also went through a slight mode collapse on the CelebA and Cartoon face datasets. The images of faces generated by it were vague and had strange details. The TL-GAN solved the mode collapse problem on the small datasets and guaranteed sample diversity. Moreover, the generator network quickly produced the images. It yielded quality images of faces.

2) *Comparison of convergence*: Loss convergence on the Cartoon dataset is shown in Figure 5. Loss of the DCGAN in the generator network increased, probably owing to the mode collapse caused by a too small dataset. The discriminator loss of the DCGAN also oscillated, and it could not differentiate real from the generated samples accurately. The loss of the generator and discriminator networks oscillated during the training of the LSGAN, which experienced a moderate mode collapse. Its convergence was also slow. The loss of the WGAN in the generator and discriminator networks tended to be normal, but was slower than that in our method and resulted in unclear samples generated after training. Gradient

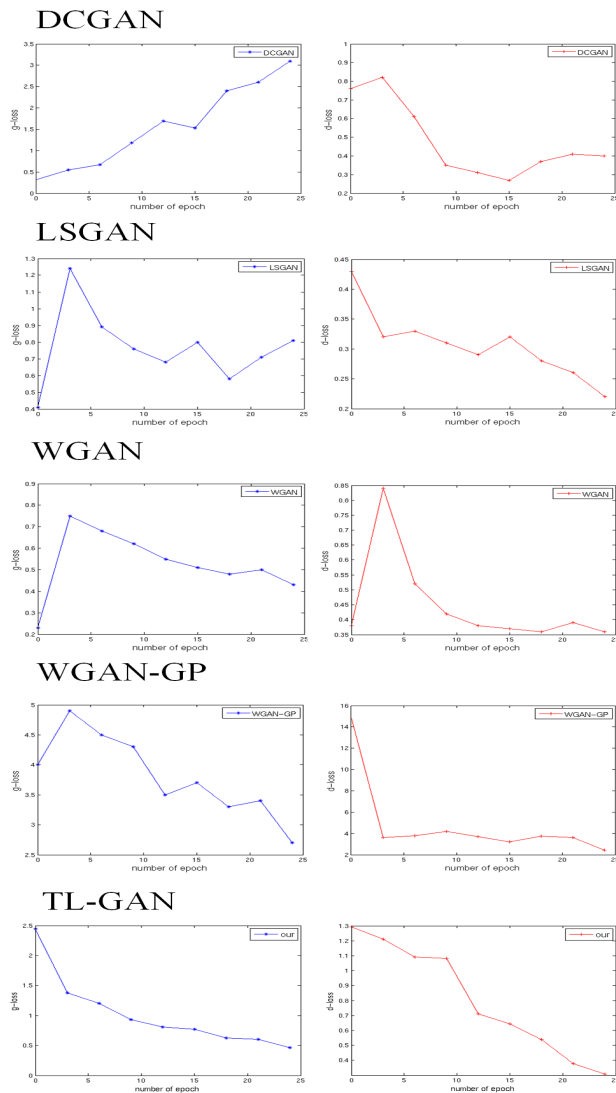


Fig. 5. Loss convergence in training on the Cartoon dataset. The generator is shown at the top and the discriminator at the bottom.

penalty was introduced in the WGAN-GP, and increased the losses of the generator and discriminator networks at first. As training progressed, the losses decreased. But after training, the results were still not good, and the generated samples were thus insufficiently clear. The results on even the Cartoon dataset were unstable, as a single sample constantly changed and there was no fixed shape to the images generated. The TL-GAN converged more quickly and was more stable than the other methods. It also produced more perceptually realistic images.

V. CONCLUSION

In this paper, we highlighted the problem of mode collapse on small image datasets and proposed generative adversarial networks with transfer learning for mode collapse (TL-GAN). Experimental results show that our method substantially outperforms state-of-the-art approaches, and samples generated

by it were close to real images. It can also ensure sample diversity. Compared with the other methods, the loss convergence of our model was fast and stable. In the future, we will continue to study how to use smaller data sets, such as thousands of images. Whether there will be collapse and good stability in our model.

REFERENCES

- [1] X. Li, J. Lv, and Y. Zhang, "Manifold alignment based on sparse local structures of more corresponding pairs," *IJCAI '13 Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pp. 2862–2868, 2013.
- [2] —, "An efficient representation-based method for boundary point and outlier detection," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2016.
- [3] —, "Outlier detection using structural scores in a high-dimensional space," *IEEE Transactions on Cybernetics*, pp. 1–9, 2016.
- [4] X. Wu, X. Li, J. He, X. Wu, and I. Mumtaz, "Generative adversarial networks with enhanced symmetric residual units for single image super-resolution," *International Conference on Multimedia Modeling*, pp. 483–494, 2019.
- [5] C. Luo, X. Li, L. Wang, J. He, D. Li, and J. Zhou, "How does the data set affect cnn-based image classification performance?" *2018 5th International Conference on Systems and Informatics (ICSAI)*, pp. 361–366, 2018.
- [6] P. Isola, J. Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5967–5976, 2016.
- [7] Y. Taigman, A. Polyak, and L. Wolf, "Unsupervised cross-domain image generation," *Computer Science - Computer Vision and Pattern Recognition*, 2016.
- [8] J. Y. Zhu, P. Krahenbuhl, E. Shechtman, and A. A. Efros, *Generative Visual Manipulation on the Natural Image Manifold*. Springer International Publishing, 2016.
- [9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," 2014, pp. 2672–2680.
- [10] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *Computer Science*, 2015.
- [11] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks," *Computer Science - Computer Vision and Pattern Recognition*, 2016.
- [12] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *Statistics - Machine Learning Computer Science - Learning*, 2017.
- [13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," 2017.
- [14] H. Wu, S. Zheng, J. Zhang, and K. Huang, "Gp-gan: Towards realistic high-resolution image blending," 2017.
- [15] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2017, pp. 2242–2251.
- [16] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," 2015.
- [17] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 105–114, 2016.
- [18] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," 2014, pp. 1717–1724.
- [19] Z. HE, "The cartoon face data set," 2018. [Online]. Available: <https://zhuanlan.zhihu.com/p/24767059>

Piecewise Aggregation for HMM fitting. A pre-fitting model for seamless integration with time series data.

Joaquim Assunção [†], Jean-Marc Vincent ^{*}, Paulo Fernandes [‡]

[†] UFSM - Department of Applied Computing - Santa Maria, Brazil

^{*} Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France

[‡] Roberts Wesleyan College – Rochester, NY – USA

[†] joaquim@inf.ufsm.br, ^{*} jean-marc.Vincent@imag.fr, [‡] fernandes_paulo@roberts.edu

Abstract

Broadly used and applied in many domains, Hidden Markov Models are a well established formalism, both in computer science and statistics. Among other reasons, they owe their popularity to a fast fitting method, *i.e.*, the Baum-Welch algorithm, allowing to adjust models to a variety of input data. Using expectation and maximization phases, BW assures an increase to the model likelihood at every iteration. Yet, to initialize the sequence of expectation-maximization (EM) steps, it is a standard procedure to start the BW algorithm from randomly generated values. We propose a, simple and fast, deterministic pre-fitting approach which derives the BW's initial values directly from the input data.

1 Introduction

Due to their relative simplicity and power to represent complex systems, Hidden Markov Models (HMMs) are one of the most widely used stochastic formalism for time series [31]. HMMs owe their flexibility and their ease of application to a well-developed methodological framework, including a model fitting algorithm. The so-called Baum-Welch algorithm (BW) can be considered a special case of the Expectation-Maximization (EM) algorithm (Section 2). In essence, it derives the maximum likelihood estimates for the parameters of a given model, *i.e.*, the best fit for the input data in the sense that the probability to observe the data given the model parameters is maximal. This is achieved by an iterative procedure guaranteed to increase the value of the likelihood at each iteration and such procedure finds a local maximum; therefore, offering a model solution even when the likelihood is untraceable or too costly to maximize directly. However, as well as EM algorithms in general, BW tends to be sensitive to its input parameters [6][9].

Several extensions have been suggested to overcome the flaws of EM [9][18][13][29]. They are based on combinations of algorithms and techniques such as classification, randomization or more complex stochastic additions. Others,

focused on improving the BW, have different approaches performing changes within the algorithm, which can also deal with possible convergence problems [4][26][16] [20].

Several extensions have been suggested to overcome the flaws of EM [9][18][13][29]. They are based on combinations of algorithms and techniques such as classification, randomization or more complex stochastic additions. Others, focused on improving the BW, have different approaches performing changes within the algorithm, which can also deal with possible convergence problems [4][26][16] [20].

Our solution focuses on the initialization only. It keeps the BW's structure and adds a pre-fitting deterministic step, which by avoiding bad initial parameters tends to obtain higher likelihood values in the first iteration, thus reducing the number of iterations needed to find a local maximum, which leads to a fast model fitting (Section 3). Although the difference is minimal, the possibility of deriving the initial values directly from the input data can be interesting for applying these models into different scenarios.

Since an HMM is usually used having serial data as its input [31], the idea is to use a deterministic discretization technique for time series prior to the actual model fitting. From this approximate description of the original observations, we then derive initial parameters which are fed into BW. These parameters are, in general, reasonably close to a local maximum. Thus, the combination of the parameter's selection step, in combination with the BW algorithm, is more likely to reduce the number of iterations to maximize the parameters, therefore leading to a local maximum likelihood faster than the most traditional approach, which is using random numbers to initialize the parameters.

Our approach is based on a Piecewise Aggregate Approximation (PAA) technique, which is used in the algorithm (Figure 1, Piecewise Expectation). The use of PAA with EM, Piecewise Aggregation EM (PAEM), has advantages regarding fitting speed and practicality due to its simpler set of parameters.

[—]*DOI reference number: 10.18293/SEKE2019-185

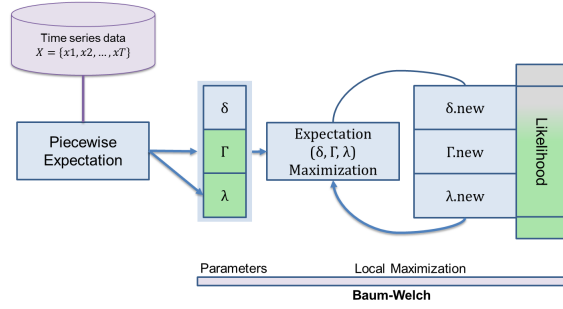


Figure 1: PAEM, schematic representation

Finally, we performed experiments and measures to compare the traditional use of BW against PAEM (Section 5). The main contribution of our approach is to reduce the time needed to fit HMM models. However, this is performed with a pre-fitting without modifying the original algorithm, which is a different approach than the others (Section 2.1). This pre-fitting can also be seamlessly used on time series data, reducing the modeling time.

2 Baum-Welch

The Baum-Welch algorithm (BW) is a version of EM algorithm for HMMs [5]. The goal is to estimate the parameters of an HMM given an input data [31], in the statistics literature, commonly described as observations. This goal implies that unlike a manual fitting approach, the initial distribution δ , the transition probability matrix Γ and the emission probabilities λ , are not estimated by the modeler observations but, automatically, by the model itself.

The BW algorithm works iteratively by successive maximizing local approximations of the likelihood function. It is guaranteed to maximize the likelihood at each iteration. EM alternates between two steps. The E-step computes the conditional expectation of the hidden states given the observations and the model current parameters, Γ , δ and λ . These computations are based on the complete data log-likelihood, which is basically the natural logarithm of the likelihood function to avoid the underflow problem [14]. In the M-step, the expectations are maximized according to its parameters.

2.1 Variants and derivations There are several variants and extension for the standard EM algorithm, also described as Generic EM (GEM). This section concisely shows an extensive literature review for the EM variants and derivations. We briefly classify the GEM-based algorithm, in order to establish their relations and differences compared to our approach.

We can globally classify these variants into deterministic and stochastic versions. Among the deterministic versions, Classification EM (CEM) [9], Accelerated EM (AEM) [18], Aitken's acceleration (AA)[13], [25], Expectation

tation Conditional Maximization (ECM)[29], ECM Either (ECME) [23], Space-Alternating Generalized EM (SAGE) [15], Parameter-Expanded EM (PX-EM) [24].

The stochastic versions include Stochastic EM (SEM) [8], Stochastic Approximation type EM (SAEM) [8], Data Augmentation algorithm (DA) [27] and Monte Carlo EM (MCEM) [30]. Although many of them are focused on Gaussian mixture models, all these variants have slightly different approaches to solving slightly different problems. A common problem is the EM step sensitiveness to the initial parameters [6]. Bad initial parameters will lead to more EM steps (iterations), which are necessary to find the local maximum likelihood.

All these GEM-based algorithms have in common the use of an iterative MLE or Recursive MLE (RMLE). However, not all fitting algorithms are based on the MLE. Some are based on Minimum Model Divergence (MMD) and Minimum Prediction Error (MPE), which can be extended to Recursive Prediction Error (RPE) as a general recursive stochastic algorithm [3]. MMD, in few words, can be described as a combination of MLE and the minimization of parameter's divergence using entropy measures. Also, Minimum Prediction Error (MPE) which consists of measuring an HMM error output prediction and provide an updated estimation for the HMM parameters [11]. Among the algorithms, using MPE we can emphasize Collings *et al.*, [11] and LeGland and Mevel [21]. Using MMD we can emphasize Garg and Warmuth [17].

Despite the uses of MMD and MPE, we focus on the classic MLE, which is commonly used for HMMs. So far, works based on MLE are the following: [4][26][16][20][7]. However, our approach does not intend to create an entirely new algorithm nor improve it within itself. Instead, we perform a pre-fitting to avoid the BW sensitiveness, an approach used by some EM variants. Also, we do not intend to create an optimal algorithm, but a better version of the traditional BW, which aims to be a practical option that does not suffer from the same flaws of a GEM, which as well as BW, is strongly dependent on its initial parameters [9]. Therefore, the convergence time is directly dependent on how good are these initial conditions.

3 PAEM

The efficiency of fitting HMMs can be improved by combining the EM algorithm with data mining techniques such as classification methods or the K-means algorithm [9][31]. These techniques are used to choose the initial parameters intelligently, thus reducing the impact of the EM/BW sensitiveness to them.

As showed in the Section 2.1, there are many algorithms which have been derived from the generic EM. However, they are based on different techniques and adapted for different situations. Here, we adapt and combine a Piecewise

Aggregation technique for time series to represent the original observations and perform a pre-fitting for the BW algorithm, calling this extension the Piecewise Aggregation EM (PAEM). PAEM profits from the simplicity of the PAA method and the dynamism of a SAX [22] inspired method, which can be applied for different kinds of distributions concerning time series. These characteristics allow us to derive meaningful initial parameters by a fast approximation of the data, avoiding failing on dimensionality problems, such as a fail to converge, which can be given by higher dimensions; or imprecise representations, which can be lead by a strong dimensionality reduction.

PAEM's initial approximation enhances the initial parameters Γ and λ , making them close to the global maximum, which leads to a faster fitting compared to the traditional random initialization. This is due to the need of a single initialization to maximize the parameters and to a first better fitting, which reduces the sensitiveness effect and tend do avoid EM iterations. Fig. 1 illustrates the general idea: a pre-fitting in a phase called piecewise expectation prior to the traditional BW. Two of three parameters are previously updated, Γ and λ , which together with the steady state, stored in δ , tends to lead to a first better likelihood. The piecewise expectation cost is $\vartheta(T)$, which is lower than forward-backward procedures $\vartheta(N^2T)$, briefly described in the previous section. Therefore, once a pre-fitting saves one iteration, the final computational cost should be lower.

3.1 Piecewise Expectation Piecewise Aggregate Approximation (PAA) is a technique to reduce data dimensionality through discretization. It has been widely applied in the context of time series analysis. Despite being simple and intuitive, PAA has been shown to be as powerful as more sophisticated dimensionality reduction techniques such as Discrete Fourier Transform [1], Discrete Wavelet Transform [10], Singular Value Decomposition [19].

To perform dimensionality reduction, PAA creates a discrete version of the original TS in w blocks. These blocks are usually a division of the length of the TS. In our case, the faster mapping characteristic is especially attractive. Since we intend to reduce the total time necessary to fit a model, more robust approaches might be too costly for a pre-fitting procedure.

Given a time series S with length n , $PAA(S)$ is defined as a sequence $PAA(S) = \{\mu(B_1), \mu(B_2), \dots, \mu(B_w)\}$, where μ is the mean, w is the maximum number of blocks and B_i is a block in the index i , being $(1 \leq i \leq w)$. The mean of a block is given by the Equation 3.1. If the division n/w results in a float number, the result is truncated and another block is made of the remaining part of the series.

$$(3.1) \quad \mu(B_i) = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} S_j$$

Despite its simplicity, PAA is enough to start an appropriated representation of a given series. We use it as a mapping to set up variables and then calculate Γ through MLE. Other PAA advantage is that it has only one parameter and due to it being mainly used to reduce a time series dimensionality, it is not critical for our problem. Therefore, given a set of observations $X = (x_1, x_2, \dots, x_T)$, PAEM only needs one parameter, the model number of states n . It starts by getting a sequence of symbolic values ϕ , with range n , that better describes X . The total number of Blocks is equal to $w = T/m$. Thus, we identify the approximation sequence with $\phi = (\mu(B_1), \mu(B_2), \dots, \mu(B_w))$.

The number of states n defines the division of the values $\mu(B_i)$. This generates the λ values of the HMM. Thus, vector ϕ has a sequence of w elements composed by n symbols. For instance, in $\phi = \{1, 2, 3, 1, 2, 3, 2, 2\}$, $w = 8$ and $n = 3$. Now, we can extend the whole approximation process to a set of equations.

Vector ϕ is directly used to get the probabilities and set the transition probability matrix, Γ , which together with λ are the two necessary parameters for an HMM model since δ can be initiated *null* and then filled with the steady state. Due to this solution be directly related with PAA, we call this part of the algorithm, "Piecewise Expectation".

To generate Γ , we use the elements of $\phi^{(t)}$, $1 \leq t \leq w$ and $1 \leq i, j \leq n$. The non-normalized matrix Γ is filled by the cumulative sum of the probability to find an element $\phi_j^{(t)}$ just after an element $\phi_i^{(t)}$.

$$(3.2) \quad \Gamma_{(i,j)} = \sum_{t=2}^T P(\phi_j^{(t)} | \phi_i^{(t-1)})$$

Different than Γ , the elements of λ are directly extracted from the piecewise approximation procedure. The generated values are actually close to the BW's ones. This small distance between the pre-fitted and the pos-fitted value is constantly observed in our experiments.

4 Experiments and Results

Our experiments aim to measure and compare how fast the local maximum likelihood is achieved through BW and PAEM. In other words, to prove the efficiency of our algorithm in finding the best model fit. To do so, we used randomly generated and randomly chosen time series from Data Market [12]. We separated our tests in three steps. First, to compare the generated maximum likelihood from different initialization of BW against the PAEM approach. Second, to detect the number of necessary executions to achieve the global maximum likelihood; consequently, how long it takes to achieve the global maximum likelihood. Third, a direct measurement of the user for PAEM vs BW.

Our tests followed the hypothesis that a human operator begins with no knowledge about the dataset. In other words,

we consider no previous data mining or machine learning techniques have been performed. For the last two sets of test (Section 4.2 and 4.3), the BW initialization followed the standard strategy [9], it was performed through random normalized numbers to all parameters. We used 12 different time series for models ranging from 2 to 4 states. Regarding these 36 tests, for each BW execution, 50 different seeds were used. To avoid outliers, the best and the worst 5 were taken out. From these 40, we used the best, the worst and the average measurements to compare against PAEM.

4.1 Likelihood Prior to the user time and iteration tests, we compared the fitness of BW and PAEM with only one initialization. Since the Expectation-Maximization part is the same, if the BW parameters are not equiprobable they should reach the same likelihood. Otherwise, if BW is inferior, it means that not all randomized parameters are good as an input. If PAEM is inferior, it means that the pre-fitting fails. Table 1 shows this experiment with BW and PAEM, where * means that we used an equiprobable λ to initialize the BW. In fact, if the BW's parameters values are not equiprobable, it tends to converge to a maximum likelihood. The problem usually happens when an equiprobable λ or Γ is given as an input, which is trivial to avoid.

Table 1: Experimental model measurements using random numbers as parameter initialization.

Model	# states	mLLk	AIC	BIC
BW (Equip. λ)	2	1268.92	2547.84	2560.86
BW	2	635.32	1280.65	1293.67
PAEM	2	635.32	1280.65	1293.67
BW (Equip. λ)	3	1268.92	2559.84	2588.50
BW	3	510.22	1042.44	1071.09
PAEM	3	510.22	1042.44	1071.09
BW (Equip. λ)	4	1268.92	2575.84	2625.34
BW	4	471.82	981.65	1031.15
PAEM	4	471.82	981.65	1031.15

Although an equiprobable λ suggests a bad fitting, this is not true for all scenarios. Despite a tiny improvement, in some cases, an equiprobable λ retrieved a better likelihood. For the other datasets, a similar phenomenon occurred in some models with more than 3 states. This suggests that a simple condition to avoid an equiprobable parameter may not be a good solution.

Considering one decimal precision, PAEM reaches a better likelihood in 3 cases against 2 from the pure BW. Table 2 shows these cases. For all the 36 experiments, PAEM was better in 17 occurrences against 19 of the pure BW. However the difference in the vast majority of these cases lies in a nth decimal precision, which can be seen in Table 2, it represents a negligible probability.

4.2 Iterations As in the previous section, we started by checking our hypothesis through experiments using 50 dif-

Table 2: -Log-Likelihood comparison, cases where the difference exceeds a precision of one float point.

BW	PAEM	$\Delta\%$	
210.13	209.71	0.01%	favorable to PAEM
740.52	697.38	5.80%	favorable to PAEM
773.44	718.20	7.10%	favorable to PAEM
471.82	489.84	3.60%	favorable to BW
321.55	322.99	0.40%	favorable to BW

ferent seeds to BW, excluding the best and the worst 5. From the 40 remaining we collected the best, the average, and the worst case concerning the BW initialization and its number of iterations to reach a convergence. As PAEM generates the parameters through a deterministic technique, it only needs one initialization. Figure 2 shows the average scenario. The other scenarios have a similar behavior.

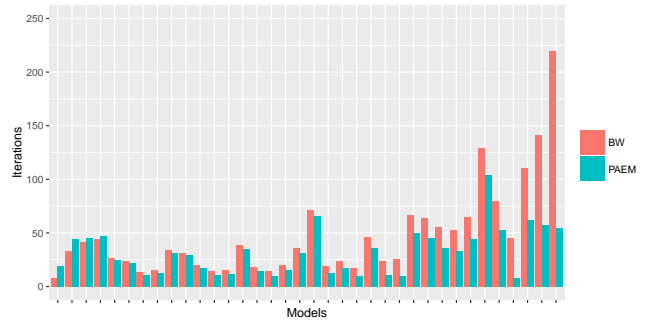


Figure 2: Average scenario for the required number of iterations to find a convergence. Experiments organized according to the models more favorable to BW (left) to the ones more favorable to PAEM (right).

In these pictures, we can clearly see PAEM requiring fewer iterations to find a convergence (right side), while just in a few cases, the random parameters outperformed PAEM (left side). Furthermore, these are retrieved from the experiments described in the Section 4.1, which shows an equivalent likelihood, between BW and PAEM, for 87% of the cases. Also, the far most significant scenario which PAEM performed poorly, loses with a difference of 3.6% (Table 2, BW=471.8).

Concerning all the results for the ordinary BW; 40 seeds for all the 12 series and the 2, 3, and 4 states model; the random values for BW got an average of 47.4 iterations against 25 from PAEM's. This shows a significant improvement for the initial parameters quality against the traditional random approach. Furthermore, in the vast majority of the tests, PAEM found a convergence with fewer iterations (Figure 2).

4.3 User Time Since both, BW and PAEM, tend to converge to the same likelihood and the cost to randomize values to BW is trivial, the real advantage of PAEM lies on a faster

convergence, which is given by fewer iterations derived by a better likelihood at the first iteration.

We performed time measurements to see how fast each procedure is in relation with BW. Although the running time is highly correlated with the number of EM iterations, a lower running time is the final goal, therefore, a more precise measure regarding the time actually used by BW and PAEM. In a standard machine, Intel i5, 2.3GHz, 8GB, a four states model had an average time of 0.34 seconds running with PAEM and 1.17 seconds running with BW. This difference is directly linked with the number of iterations. As described in the previous sections, in most cases, PAEM's pre-fitting tends to avoid at least one iteration of the forward-backward procedure, which costs $\vartheta(N^2T)$, which is more than PAA $\vartheta(T)$.

The user time spent, from both, had a strong correlation with their number of iterations. Specifically, BW had a correlation average of 93% and PAEM 76%. Which can be explained by the different seeds in BW and the lack of a precise control considering an ordinary machine running other applications. Also, PAEM's pre-fitting has a fixed running time for series with the same length, which has a different impact according to the series number of iterations necessary to find a convergence.

Now, considering the average scenario, we look for each of the 36 experiments. Thus, the radar showed by Figure 3, illustrates the total time spent in relation to the average scenario for each time series. From this figure, we can clearly see the time percentage difference from each technique and for each time series. This plot shows the overall better performance of PAEM, failing in just 6 cases, which are the time series 1, 10, 16, 21, 24, and 26. However, a difference in the case 26 is meaningless since the difference is 0.0004 in favor of BW.

Among these time series, a critical poor performance was achieved on the time series #1 and #10, which happens to be the shortest time series in the experiment. Considering BW and PAEM, respectively, for the first time series, considering an average case, it required 0.026 and 0.110 seconds. Time series X10, required, respectively, 0.025 and 0.064 seconds.

Considering larger models, we can verify that PAEM performed better, in average, for any number of states less or equal to 22. Further tests are required for larger models. Finally, we emphasize that, our code was extended from [31] and they do not have focus on performance. Therefore, the user time is far from optimal and the difference might be much less than the observed in our experiments.

5 Discussion

Through exhaustive tests, with different series, we found PAEM to be faster than BW with its traditional stochastic initialization. Its performance is due to, usually, fewer itera-

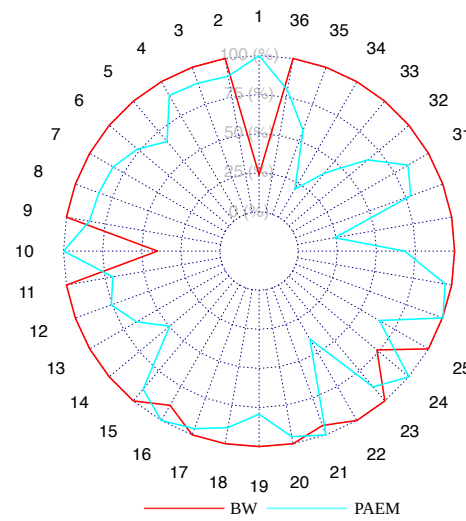


Figure 3: Radar chart showing the slowest process taking 100% of the time. An average scenario considering the user time.

tions in the EM procedure. In fact, as shown in Figure 2, in the vast majority of executions, the convergence is achieved using a fewer number of iterations than the pure BW. Furthermore, the time needed to the piecewise approximation is far smaller than one EM iteration.

However, it is important to emphasize the overall better performance and that PAEM does not aim to be an optimal solution. We focus on a simple alternative to the initial and usual randomization of parameters. Although there are other techniques that improve the original BW, PAEM lies in a simple initialization that is fast and easy to implement, making it a suitable alternative to performing HMM fitting. In fact, there are cases where authors related a faster solution using simpler MLEs [2]. Other techniques such as the Levenberg-Marquardt algorithm, can be used to maximize directly the likelihood, which can be faster than EM approaches [28].

For the future improvements, we shall focus on measuring the initial likelihood and the user time according to different kinds of data and distributions. PAEM is based on a simple Piecewise Aggregation technique. It may have a better global performance if a more advanced technique is used instead of Piecewise Aggregation. For this reason, we do not focus on measure the impact of w in its parameters. In a future work, we can focus on comparisons, such as the impact of different values of w and more robust techniques, like SAX [22] and its derivations. However, the time spent to pre-process the data must be lower than the original one. Otherwise, the overall performance might decrease. Another important test is to detect how efficient PAEM scales regarding models with a different number of states.

References

- [1] R. AGRAWAL, C. FALOUTSOS, AND A. N. SWAMI, *Efficient similarity search in sequence databases*, in Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms, FODO '93, London, UK, UK, 1993, Springer-Verlag, pp. 69–84.
- [2] R. M. ALTMAN AND A. J. PETKAU, *Application of hidden markov models to multiple sclerosis lesion count data*, *Statistics in Medicine*, 24 (2005), pp. 2335–2344.
- [3] A. ARAPOSTATHIS AND S. I. MARCUS, *Analysis of an identification algorithm arising in the adaptive estimation of markov chains*, *Mathematics of Control, Signals and Systems*, 3 (1990), pp. 1–29.
- [4] P. BALDI AND Y. CHAUVIN, *Smooth on-line learning algorithms for hidden markov models*, *Neural Comput.*, 6 (1994), pp. 307–318.
- [5] L. E. BAUM, T. PETRIE, G. SOULES, AND N. WEISS, *A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains*, *The Annals of Mathematical Statistics*, 41 (1970), pp. 164–171.
- [6] C. BIERNACKI, G. CELEUX, AND G. GOVAERT, *Choosing starting values for the {EM} algorithm for getting the highest likelihood in multivariate gaussian mixture models*, *Computational Statistics & Data Analysis*, 41 (2003), pp. 561 – 575. Recent Developments in Mixture Model.
- [7] O. CAPPE, V. BUCHOUX, AND E. MOULINES, *Quasi-newton method for maximum likelihood estimation of hidden markov models*, in *Acoustics, Speech and Signal Processing*, 1998. Proceedings of the 1998 IEEE International Conference on, vol. 4, May 1998, pp. 2265–2268 vol.4.
- [8] G. CELEUX, D. CHAUVEAU, AND J. DIEBOLT, *On Stochastic Versions of the EM Algorithm*, Research Report RR-2514, 1995.
- [9] G. CELEUX AND G. GOVAERT, *A classification {EM} algorithm for clustering and two stochastic versions*, *Computational Statistics & Data Analysis*, 14 (1992), pp. 315 – 332.
- [10] K.-P. CHAN AND A.-C. FU, *Efficient time series matching by wavelets*, in *Data Engineering*, 1999. Proceedings., 15th International Conference on, Mar 1999, pp. 126–133.
- [11] I. B. COLLINGS, V. KRISHNAMURTHY, AND J. B. MOORE, *On-line identification of hidden markov models via recursive prediction error techniques*, *IEEE Transactions on Signal Processing*, 42 (1994), pp. 3535–3539.
- [12] DATAMARKET!, *The open portal to thousands of datasets from leading global providers*. <http://datamarket.com/>, 2013.
- [13] A. P. DEMPSTER, N. M. LAIRD, AND D. B. RUBIN, *Maximum likelihood from incomplete data via the em algorithm*, *Journal of the Royal Statistical Society, series B*, 39 (1977), pp. 1–38.
- [14] R. DURBIN, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press, 1998.
- [15] J. FESSLER AND A. HERO, *Space-alternating generalized expectation-maximization algorithm*, *Signal Processing, IEEE Transactions on*, 42 (1994), pp. 2664–2677.
- [16] G. FLOREZ-LARRAHONDO, S. BRIDGES, AND E. A. HANSEN, *Incremental estimation of discrete hidden markov models based on a new backward procedure*, in Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2, AAAI'05, AAAI Press, 2005, pp. 758–763.
- [17] A. GARG AND M. K. WARMUTH, *Inline updates for hmms.*, in *Interspeech, ISCA*, 2003.
- [18] M. JAMSHIDIAN AND R. I. JENNRICH, *Conjugate gradient acceleration of the em algorithm*, *Journal of the American Statistical Association*, 88 (1993), pp. 221 – 228.
- [19] F. KORN, H. V. JAGADISH, AND C. FALOUTSOS, *Efficiently supporting ad hoc queries in large datasets of time sequences*, in Proceedings of the 1997 ACM SIGMOD international conference on Management of data, SIGMOD 97, New York, NY, USA, 1997, ACM, pp. 289–300.
- [20] V. KRISHNAMURTHY AND J. B. MOORE, *On-line estimation of hidden markov model parameters based on the kullback-leibler information measure*, *IEEE Transactions on Signal Processing*, 41 (1993), pp. 2557–2573.
- [21] F. LEGLAND AND L. MEVEL, *Recursive identification of hmms with observations in a finite set*, in *Decision and Control*, 1995., Proceedings of the 34th IEEE Conference on, vol. 1, Dec 1995, pp. 216–221 vol.1.
- [22] J. LIN, E. KEOGH, S. LONARDI, AND B. CHIU, *A symbolic representation of time series, with implications for streaming algorithms*, in Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, DMKD '03, New York, NY, USA, 2003, ACM, pp. 2–11.
- [23] C. LIU AND D. B. RUBIN, *The ecme algorithm: A simple extension of em and ecm with faster monotone convergence*, *Biometrika*, 81 (1994), pp. pp. 633–648.
- [24] C. LIU, D. B. RUBIN, AND Y. N. WU, *Parameter expansion to accelerate em: The px-em algorithm*, *Biometrika*, 85 (1998), pp. pp. 755–770.
- [25] R. SALAKHUTDINOV, S. ROWEIS, AND Z. GHAHRAMANI, *Expectation-Conjugate Gradient: An Alternative to EM*.
- [26] S. SIVAPRAKASAM AND K. S. SHANMUGAN, *A forward-only recursion based hmm for modeling burst errors in digital channels*, in *Global Telecommunications Conference*, 1995. GLOBECOM '95., IEEE, vol. 2, Nov 1995, pp. 1054–1058 vol.2.
- [27] M. A. TANNER AND W. H. WONG, *The calculation of posterior distributions by data augmentation*, *Journal of the American Statistical Association*, 82 (1987), pp. pp. 528–540.
- [28] R. TURNER, *Direct maximization of the likelihood of a hidden markov model*, *Computational Statistics & Data Analysis*, 52 (2008), pp. 4147 – 4160.
- [29] D. A. VAN DYK, X.-L. MENG, AND D. B. RUBIN, *Maximum likelihood estimation via the ecm algorithm: computing the asymptotic variance*, *Statistica Sinica*, (1995), pp. 55–75.
- [30] G. C. G. WEI AND M. A. TANNER, *A monte carlo implementation of the em algorithm and the poor man's data augmentation algorithms*, *Journal of the American Statistical Association*, 85 (1990), pp. pp. 699–704.
- [31] W. ZUCCHINI AND I. MACDONALD, *Hidden Markov Models for Time Series: An Introduction Using R*, Chapman & Hall/CRC Monographs on Statistics & Applied Probability, Taylor & Francis, 2009.

How do Practitioners Manage Decision Knowledge during Continuous Software Engineering?

Anja Kleebaum¹, Jan Ole Johanssen², Barbara Paech¹, and Bernd Bruegge²

¹Heidelberg University, Heidelberg, Germany, {kleebaum, paech}@informatik.uni-heidelberg.de

²Technical University of Munich, Munich, Germany, {jan.johanssen, bruegge}@in.tum.de

Abstract—Continuous software engineering (CSE) is an agile process that supports lightweight, flexible, and rapid software development. Decision-making is crucial for CSE, and developers need to know the decisions made and the related rationale to evolve the software. This knowledge is called decision knowledge. The management of decision knowledge in CSE environments remains unexplored. The agile manifesto suggests to value working software over comprehensive documentation as well as individuals and interactions over processes and tools. What does this mean for the documentation, exploitation, and sharing of decision knowledge? We report on results from an interview study with 24 practitioners from 17 companies on how decision knowledge is managed during CSE. The practitioners mainly capture decision knowledge in an informal way, for example, in natural language discussions. Wiki and issue tracking systems represent the preferred medium to preserve decision knowledge. Mentioned benefits are an improved decision-making process, accountability, knowledge sharing, and reuse. However, the exploitation of the captured decision knowledge remains partly unclear.

Index Terms—Decision knowledge, rationale, interview study, continuous software engineering, agile software development.

I. INTRODUCTION

Continuous software engineering (CSE) is an agile process that supports lightweight, flexible, and rapid software development [1]. This process is intertwined with ongoing issue-solving and decision-making. For example, developers make decisions regarding the software development process, existence and non-existence of software artifacts, or the software quality [2]. Developers need decision knowledge, i.e., knowledge about the decisions and their rationale, to evolve the software. Rationale covers the justification behind decisions.

CSE offers opportunities and bears challenges for the management of decision knowledge. The opportunities are that developers already document decision knowledge in documentation locations such as commit messages, issue comments, or pull requests during established development practices, e.g., when committing code changes. Thus, the documentation of decision knowledge during CSE is non-intrusive in comparison to using a separate tool. This might help to overcome the capture problem that is often mentioned in articles about decision management [3]. The challenges are that the documented decision knowledge might be hard to access and exploit for developers since it is distributed and might be not formalized. Further, decisions can rapidly be changed which might lead to an inconsistent and outdated documentation. It is unexplored how decision knowledge is managed during CSE in practice.

We conducted a semi-structured interview study with practitioners from 17 companies using CSE. We already reported on the state-of-the-practice of CSE in these companies [4], [5]. In this paper, we report on how the companies manage decision knowledge. We contribute insights on which types of decisions practitioners think are important to capture, how they capture decision knowledge, what benefits they see in capturing decision knowledge, how they share decision knowledge, and how they deal with change. These insights should help us to reach our overall goal to support software evolution with decision and usage knowledge in CSE [6], [7].

The remainder of this paper is structured as follows. In Section II, we present our research questions, the procedure of the interview study, and descriptive data of the interviewees. In Section III, we answer the research questions. Section IV discusses this work and Section V lists threats to validity. Section VI presents related work on decision knowledge management in practice. Section VII concludes the paper.

II. RESEARCH METHOD

In the following, we describe our research questions, the interview study, and data about the study participants.

A. Research Questions

We focus on four research questions, each refined by sub-questions that we asked the practitioners during the interviews.

RQ1: *Which decisions are captured, why, and how?* This research question investigates approaches to explicitly capture decisions and rationale during CSE within companies. Sub-questions are: Which types of decisions do practitioners capture? Where do practitioners capture the decisions, with which techniques and tools? Do practitioners link decisions and rationale to other software artifacts and if so, how? How do practitioners preserve the evolution history of decisions? When and how often do practitioners capture decisions? Why do practitioners capture decisions, i.e., what are the benefits and what do they do with the captured decisions?

RQ2: *Which important decisions are not captured by practitioners, why not?* This research question aims to find types of decisions that remain implicit and reasons why they are not captured. Sub-questions are: Which important decisions do practitioners not capture during CSE? Why do practitioners not capture these decisions? What would be the benefits if practitioners captured these decisions?

RQ3: *How do practitioners share decision knowledge?* We want to investigate how practitioners share decision knowledge during CSE, with these sub-questions: What are the knowledge sources from which practitioners retrieve necessary information for decisions that are not captured? How do practitioners share knowledge to avoid knowledge vaporization?

RQ4: *How do practitioners deal with changing decisions?* We aim to investigate on how practitioners deal with change, with the sub-question: How do practitioners identify parts of the system that are affected by new or changed decisions?

B. Interview Study

In the following, we summarize the realization of the interview study that is described in more detail in [4], [5].

1) *Procedure:* We performed a semi-structured interview study [8], which is a survey study and thus a means to perform a field study [9], [10]. We separated the study into a *design and planning*, *data collection*, and *data analysis* phase.

During the *design and planning* phase, we prepared a questionnaire. Its first part addresses the practitioners' background and working context. Furthermore, it contained interview questions as listed below the respective research question in Section II-A. The interviews also included research questions that are not addressed in this article (see [4], [5]). We contacted companies which to our knowledge apply CSE.

During the *data collection* phase, we conducted 20 interviews between April and September of 2017. The interviews were conducted either in person or via phone. The interviews took 70 minutes on average and were audio-recorded with the permission of the interviewees. We transcribed the audio recordings and sent the transcripts to the interviewees to correct misunderstandings. We guaranteed the anonymity of the practitioners by publishing only aggregated results.

In the *data analysis* phase, we analyzed the transcripts [11]. We utilized a *qualitative data analysis* software to apply two stages. During the first stage, we allocated answers to an interview question. Hereafter, we performed a fine-grained coding stage. To answer the interview questions concerning the types of decisions captured and not captured, we derived the codes from Kruchten's taxonomy [2]. This taxonomy distinguishes between existence decisions, non-existence decisions (bans), property decisions, and executive decisions. For the remaining interview questions, we identified emerging topics and coded the answers in terms of these topics. We analyzed the results quantitatively. In the case that two interviewees participated in an interview, we treated their answers as one subject.

2) *Participants:* During 20 interviews, we interviewed 24 practitioners from 17 companies. While seven of the companies provide consultancy services, ten companies develop software products. Based on their role description, we grouped the 24 practitioners into five categories: *CSE specialists* (5), e.g., a continuous deployment manager or a DevOps engineer, *developers* (6), *project managers* (6), *technical leaders* (6), and one *executive director*. On average, the practitioners have spent two years in the respective role, have an experience in IT projects of ten years, and participated in 19 IT projects.

III. RESULTS

We present results on the research questions introduced in Section II-A. Each subsection starts with a summary followed by a more detailed analysis of the research question.

A. Decisions Captured during CSE

In three interviews, the practitioners state that they do not capture decisions at all. In these cases, we skipped the interview questions for RQ1 and started with RQ2.

RQ1: *Which decisions are captured, why, and how?* Practitioners mainly capture executive and existence decisions regarding the software architecture and feature implementation. They mostly capture decisions in wiki and issue tracking systems in informal discussions and rely on techniques for establishing trace links and version control that come with these systems. Practitioners capture decisions as part of regular practices, such as code reviews and meetings. They mention improved decision-making, accountability, knowledge sharing, as well as reuse support as benefits. However, the exploitation of the decision knowledge is partly unclear.

1) *Types of Captured Decisions:* Twelve practitioners report that they capture *executive decisions*, i.e., decisions concerning the software development process, technologies, or applied tools. Such decisions impact the entire project or several projects. Similar to non-CSE environments, the executive decisions can be made by a steering committee. However, one practitioner highlights that CSE supports strongly that developers themselves are enabled to make high-level decisions. As examples, the practitioners mention to capture the decision to use a certain branching strategy or to do continuous delivery. One practitioner mentions to capture decisions on when they can consider a task as done, i.e., the definition of done, and on when a build can be deployed to the users. *Existence decisions* state that some elements will appear in the software [2]. Thirteen practitioners capture existence decisions concerning requirements, architecture, implementation, test cases, and bug reports. Six practitioners report that they capture decisions related to the elicitation, prioritization, and effort estimation of requirements for features. Eight practitioners mention that they capture architecture decisions and another nine mention that they capture decisions regarding the implementation of features, e.g., on why a class was created. *Non-existence decisions or bans* state that some elements will not appear in the software [2]. Five practitioners report to capture possible alternatives to solve a decision problem during their decision-making process. After evaluating the alternatives against criteria, they pick one alternative as the decision. The alternatives they discard are documented non-existence decisions. One practitioner reports to capture decisions regarding the prioritization of test cases and bug fixing activities based on risk assessment. *Property decisions* concern the quality of the system and can be guidelines, design rules, or constraints [2]. One practitioner provides the example that they captured the decision on how to deal with data inconsistency after they have replaced their relational database with a NoSQL database.

2) *Documentation Locations, Techniques, and Tools*: Practitioners use various documentation locations, techniques, and tools to capture decisions during CSE. Eight practitioners mention that they capture decisions in *external documents and tools* such as *Word* files, architecture design documents, or final project reports. Only one practitioner mentions to use an architecture management tool, which in their case is the *Enterprise Architect*. Thirteen practitioners report on using a *wiki system* such as *Confluence*. One practitioner mentions that they rely on template pages to capture decisions. Ten practitioners mention that they capture decisions in an *issue tracking and project management system*, such as *JIRA* or *Redmine*, as part of the issue description and its comments. One practitioner describes that they use a distinct discovery issue type to indicate that a decision needs to be made. Similarly, another practitioner mentions that they use a tag to mark those issues that contain an open decision. One practitioner highlights that in their opinion *pull requests* are the best place to capture decisions to implement features. They create feature branches for a requirement and create a pull request directly afterwards to discuss the feature implementation within the pull request. Another practitioner reports that they document decisions as part of the *code* in comments and in *code reviews*. Code reviews can be done in pull requests, issue comments, or using dedicated code review systems, such as *gerrit*. Three practitioners mention commit messages as a documentation location for decisions and another three mention informal communication systems, e. g., *chat tools* like *Slack*, and *emails*.

3) *Linked Artifacts*: None of the practitioners uses a particular technique to establish links between captured decision knowledge and software artifacts. However, practitioners mention techniques that come naturally with capturing decision knowledge in some documentation locations. For example, the practitioners report that the decisions captured in the issue tracking system can be traced to the respective issues such as user stories and also to artifacts that are linked to these issues, e. g., software components and code. In addition, they also mention that separate documents or wiki pages can be tagged; for instance, version numbers can enable traceability between decisions and software builds. Practitioners find it hard to keep the documentation of decisions and software artifacts in a consistent state. The practitioner using the architecture knowledge management tool criticizes that there are no links between the design models and the wiki system where they also capture decisions. They insert snapshots of the models into the wiki page, which they rate as highly unusable, especially when the models get changed. Another practitioner suggests to capture decisions as close to the code as possible.

4) *Evolution History of Decisions*: Similar to the linking of decision knowledge and artifacts, a preservation of the evolution history comes naturally in those *documentation locations that offer version control*, e. g., the issue tracking system. One practitioner describes that they have a technique to *mark a revised decision*; they link the revised decision with the new one rather than overwriting the revised decision. However, the practitioner admits that they never used the technique.

5) *Capturing Practices and Frequencies*: Six practitioners report that they mainly capture decisions *on demand*, e. g., when planning bigger updates. One practitioner states that they only capture a decision in case they need to discuss on it, i. e., for controversial issues. Seven practitioners mention that they capture decisions as part of *regular practices* such as code reviews, meeting, and retrospectives. The practitioner reporting about the tag to mark an open decision states that the product owner regularly filters for such tagged issues.

6) *Benefits and Exploitation*: Five practitioners mention that they document decisions since the documentation improves *decision-making*, i. e., leads to better decisions since the criteria become clearer. Eight practitioners state that they capture decisions and rationale for *accountability* reasons, e. g., as a proof on why a certain feature has been developed and to avoid misunderstandings in the future. One of them states to exploit captured decisions when recovering a former state of the software. Three practitioners state that they capture decisions for *knowledge sharing* purposes. Among them, one practitioner highlights that it is necessary to share the knowledge about where a new decision needs to be made, i. e., also to share issues. Two practitioners capture decisions in order to support *reuse* in the future to avoid duplicated work.

We asked the practitioners to rate the statement *The explicit capturing of decisions benefits our software development process* with one answer from a five point Likert scale. In thirteen interviews the practitioners rated this statement: one disagreed, three were neutral, and nine agreed (Figure 1). The practitioners who disagreed and were neutral emphasized that if the utilization of the captured knowledge was more clear, they would give a higher rating.

B. Decisions not Captured during CSE

Although some practitioners capture executive, existence, non-existence, and property decisions during CSE, others either a) do not capture the same type of decisions or b) provide other concrete examples for decisions that they do not capture.

RQ2: Which important decisions are not captured by practitioners, why not? Decisions regarding the CSE process, prioritization, alternatives that are not selected (non-existence decisions), and the underlying rationale stay implicit. Practitioners do not capture decision knowledge because they fear intrusiveness and inconsistency, miss clear use cases for exploitation as well as techniques and tools. They see a potential benefit in supporting software evolution through captured non-existence decisions and decisions for code.

1) *Types of Decisions not Captured*: Seven practitioners provide examples for *executive decisions* regarding the CSE process that they do not capture but that they think would be important to capture. They state that the decisions on the continuous integration and deployment pipeline and the respective stages, e. g., the develop, test, and production stages, stay implicit in the head of developers. In total, eleven practitioners mention that they do not capture certain *existence decisions*. Such decisions relate to features, software

The explicit capturing of decisions ...

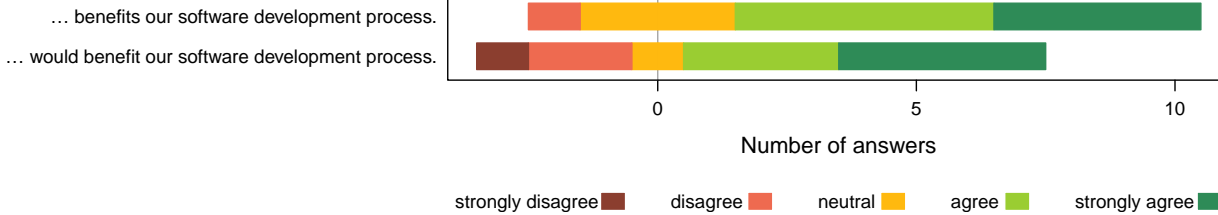


Figure 1. The practitioners' attitude towards capturing decisions (above) and towards capturing decisions that they currently do not capture (below).

architecture, implementation, and tests. For example, a practitioner reports to document application programming interfaces between microservices using *Swagger* but does not capture decisions for the design of such interfaces and the underlying rationale. The practitioners report to capture the outcome value regarding the prioritization of requirements based on cost estimation, test cases based on risk estimation, and bug fixing activities. However, the rationale is not captured, especially if it comes to reprioritization. Two practitioners report that they do not capture configuration decisions, e.g., which compiler or framework versions they use. Three practitioners criticize that they do not capture the rationale behind decisions and that they do not capture decisions on why they did not pick a certain alternative for an issue, i.e., *non-existence decisions or bans* stay implicit. Two practitioners mention that they have a common understanding of certain *property decisions*, e.g., about the coding style, but that such decisions are not documented. One practitioner provides the example that they did not capture the decision whether to use either a synchronous or an asynchronous inter-service communication between microservices. The practitioner states that these kind of decisions are made very quickly and then get reused by others, but are neither discussed nor captured.

2) *Reasons why Decisions are not Captured*: Two practitioners report that the decisions on how to deploy the software used to be captured in external documents but are no longer captured since the deployment is now automated. However, they still keep the former documents to externalize this knowledge. Four practitioners see a problem in *rapid changing decisions* that lead to outdated decisions, i.e., to inconsistency between the captured decisions and their implementation. One practitioner associates the waterfall process with capturing decision knowledge. Five practitioners report that they *lack appropriate techniques or tools* to capture decisions and rationale. Three of them state that their *process is not mature enough* to involve decision management. Six practitioners do not capture decisions because they *lack techniques for an easy retrieval and exploitation* of the captured decisions. Eight practitioners fear the *overhead and the intrusiveness* of capturing decisions and rationale. They could not spend the effort and do not have enough time. One practitioner mentions that the cost-benefit-ratio would be too high if they captured more decisions than they currently do according to the 80/20 rule. However, the practitioners admit that the extra effort could be reduced by applying better capturing techniques.

3) *Potential Benefits if Captured*: As for the captured decisions, practitioners see potential benefits in establishing accountability, improving decision-making and knowledge sharing, as well as a support of reuse and maintenance activities. They also stress that capturing decisions and rationale would support continuous learning as part of the CSE process. Two practitioners see a potential benefit in retrieving decisions and rationale for code when evolving code. In their opinion, this could *ease the understanding of code*. Three practitioners state that it would be useful for them to know about alternatives for a decision and the rationale why they were not selected during software evolution. One practitioner mentions disaster recovery as an example why knowledge sharing and capturing decisions was important.

We asked the practitioners to rate the statement *The explicit capturing of decisions would benefit our software development process* regarding the decisions that they currently do not capture. Practitioners of eleven interviews rated this statement: three disagreed, one was neutral, and seven agreed (Figure 1).

C. Sharing of Decision Knowledge during CSE

We dedicated two interview questions to address this research question.

RQ3: How do practitioners share decision knowledge?

Practitioners strongly rely on face-to-face communication, i.e., colleagues' knowledge, to recover implicit decisions. To share knowledge equally they apply techniques such as pair programming and inviting all team members as reviewers to pull requests. However, they also try to recover implicit decisions using reverse engineering.

1) *Alternative Knowledge Sources*: Six practitioners state that they try to do *reverse engineering* to recover knowledge from code and issue tracking systems. Ten practitioners mention that they *ask colleagues*, which has the disadvantage that both the inquiring person and the respondent need to interrupt their current activity. One practitioner reports that they have an *emergency mobile phone* that is carried by one knowledgeable project member for a period of time; afterwards, it is passed to the next project member. Two practitioners report that it can be hard to *scan through many emails and pull requests* to recover a decision. Thus, this decision was somehow documented but hard to retrieve. Another practitioner enforces that decisions are hard to retrieve in communication channels using the slogan "if it happens in [chat tool], it did not happen".

2) *Avoidance of Knowledge Vaporization*: The practitioners try to avoid knowledge vaporization by sharing knowledge between project members. One practitioner states that in larger teams it is both necessary to share the knowledge within and across team boundaries. Knowledge management should address both the intra- and inter-team scope. Within teams, the practitioners try to share knowledge between all members as homogeneously as possible. They strongly rely on face-to-face communication. Further, one practitioner mentions that they always *invite all team members as reviewers for pull requests* and also do *pair programming* to distribute knowledge. One practitioner states that they encourage team members to always share their notes with others, e.g., by using a wiki system, instead of “writing diaries”. Two practitioners mention to have a dedicated process to onboard new project members. Generally, practitioners state that if a project member is about to leave the company, they would have a period of time during which this person tries to share and capture their knowledge.

D. Managing Changing Decisions during CSE

Overall, we received only few responses from practitioners regarding the management of changing decisions during CSE.

RQ4: *How do practitioners deal with changing decisions?*

Practitioners use cost and risk estimation as well as prioritization before integrating changing decisions. They depend on implicit knowledge and team communication to identify parts of the system affected by new or changed decisions. They rely on automated tests to detect side and ripple effects.

None of the practitioners report about a technique or tool to identify parts of the system that are affected by new or changed decisions. One practitioner reports about their *change management* process. For a change request, the project leader needs to decide whether the change will be integrated and—if so—the developers *estimate the cost* for the change, *define a priority*, and break it down into tasks. Other practitioners emphasize the importance of *automated tests* to detect side and ripple effects as well as *risk management*. One practitioner of a consulting company criticizes that workflows often do not scale when the project and the respective team sizes increase. Change impact analysis would be especially important for larger projects, however, it is not integrated since it had not been necessary at the beginning when the project was small.

IV. DISCUSSION

In the following, we discuss the results in terms of findings, problems, and our improvement ideas.

From the results of our interview study, we cannot make a clear statement of which decisions are captured and which decisions stay implicit, since some practitioners mentioned to capture decisions that others do not capture and vice versa. Yet, the answers towards RQ1 and RQ2 provide examples of decisions practitioners consider important to be captured and for which purposes. It is interesting that many practitioners find executive decisions regarding the CSE process important to be captured. Reasons might be that CSE involves a continuous

process improvement that comes with a continuous decision-making. The CSE process contains many defined workflows that developers need to decide on and for which they need to have a common understanding [1], [4].

Our findings confirm the challenges of CSE listed in Section I: In 19 interviews, the practitioners mention that their decision capturing method needs to be improved and that it is far from being perfect. Only in one interview, a practitioner in the role of a quality manager states that they are very focused to capture decisions. The degree of formalization of decision-making and documentation in practice seems to be rather low. During the interviews, only five practitioners mention to capture alternatives for a decision, i.e., non-existence decisions. However, Kruchten states that it is very important to capture non-existence decisions as they are not visible in the software artifacts and cannot be recovered using reverse engineering [2]. Also, the underlying rationale is not captured systematically. The practitioners argue to not capture decisions since the rapid change would make them outdated soon. Further, the practitioners state that the usage of too many tools for capturing decisions can be frustrating. As reasons they list a) redundancy, i.e., they need to document knowledge in more than one tool, which means twice the effort and might result in an inconsistent documentation, and b) a workflow interruption, i.e., they have to change their working context for documentation purposes, which means intrusiveness.

Although the practitioners confirm to document decision knowledge in typical documentation locations, e.g., the issue tracking system, the opportunities of CSE for an improved decision knowledge management are not yet exhausted. The practitioners stress that the utilization of the captured decision knowledge is not clear to them and that it is not exploited in a proper way. They also highlight that they have difficulties to find and retrieve the decisions—especially if captured in informal communication channels such as *Slack*. In summary, the capturing and exploitation of decision knowledge needs to be better integrated into the daily practices of developers.

We aim to provide solution proposals for these findings applicable for practitioners [12]. In [7] and [13], we describe ideas for a *continuous decision knowledge management (ConDec)* as part of CSE and in [14] we present a dashboard for knowledge visualization. The ConDec tool support¹ integrates with existing tools to minimize the intrusiveness of the decision knowledge management, e.g., with the issue tracking system. It provides many features for capturing and exploiting decision knowledge during CSE and supports knowledge sharing and changes. For example, ConDec enables the explicit, formal capture of decision knowledge in the description and comments of *JIRA* issues, commit messages, and code comments. It does not restrict the type of decisions, i.e., developers can capture executive, existence, non-existence, and property decisions. It enables developers to view decision knowledge in relation to software artifacts such as features and code. We evaluate the ConDec tool support in agile student courses [15].

¹<https://github.com/ures-hub>

V. THREATS TO VALIDITY

We conducted the interview study from a positivist philosophical stance, i.e., we try to draw conclusions on how practitioners manage decision knowledge during CSE from the interviews. We discuss the four criteria for validity as usually done for empirical research with a positivist stance [9], [8]. A more detailed discussion can be found in [4], [5].

Construct validity focuses on whether the theoretical constructs are measured and interpreted correctly. The practitioners might have interpreted the interview questions different to what we intended. To reveal misinterpretations, we allowed them to ask questions at any time and conducted two interviews with colleagues that we discussed afterwards. We used open-ended questions to elicit as much information as possible.

Internal validity concerns whether the results we draw really follow from the data, e.g., whether there are confounding factors that influence the results. The practitioners might have provided answers that do not fully reflect their daily work, since they knew that the results would be published. We guaranteed the full anonymity of interviewees and companies to address this. The interpretation of answers might be biased by the authors' *a priori* expectations, which we addressed by coding the transcriptions and discussing the codes.

External validity addresses the generalizability of the study results. We contacted companies that we already knew, which might result in a selection bias. It is mitigated by the fact that the authors are from two universities with different industrial contacts. Interviews are subjective, since they rely on the practitioners' statements. To reduce subjectivity, we conducted 20 interviews, to acquire a wider set of opinions.

Reliability validity concerns the study's dependency on specific researchers. After we carried out coding training and checked intercoder reliability, two authors individually coded different transcripts. We addressed this threat by discussing questions during coding. In addition, a third author of this paper supervised the interview analysis.

VI. RELATED WORK

Miesbauer and Weinreich collected 120 examples of decisions made in practice using an interview study [16]. Similar to our study, they classified these decisions according to Kruchten's taxonomy [2] and list documentation locations. They also found that the majority of the decisions were existence decisions and that property decisions were rarely mentioned. In contrast to our study, they found that practitioners did not mention non-existence decisions. They did not ask for practitioners' knowledge sharing and change management practices, but list influence factors for decision-making.

Similar to our work, Furtado *et al.* explore tools, processes, and benefits of knowledge management [17]. They found *Google Drive* and email lists applied most prevalent to capture knowledge and point out the value of the informal messaging service *Slack* to improve knowledge sharing. In contrast to our study, they did not focus on decision knowledge and their results are limited to one institution only.

VII. CONCLUSION

We reported on findings from an interview study on how practitioners manage decision knowledge in CSE environments. The practitioners mainly capture decision knowledge in an informal way, in wiki and issue tracking systems. The exploitation of the captured decision knowledge is partly unclear and needs to be improved. We develop techniques and tool support for a continuous decision knowledge management.

ACKNOWLEDGEMENTS

This work was supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution (CURES project). We thank the practitioners for sharing their insights.

REFERENCES

- [1] S. Krusche and B. Bruegge, "CSEPM - A continuous software engineering process metamodel," in *3rd International Workshop on Rapid Continuous Software Engineering*, 2017, pp. 2–8.
- [2] P. Kruchten, "An ontology of architectural design decisions in software intensive systems," in *2nd Groningen Workshop on Software Variability Management*, 2004, pp. 54–61.
- [3] A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, *Rationale Management in Software Engineering*. Springer, 2006.
- [4] J. O. Johanssen, A. Kleebaum, B. Paech, and B. Bruegge, "Practitioners' eye on continuous software engineering: An interview study," in *Int. Conf. on Softw. and System Process (ICSSP)*. ACM, 2018, pp. 41–50.
- [5] —, "Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners," *Journal of Software: Evolution and Process*, 2019.
- [6] J. O. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech, "Towards a systematic approach to integrate usage and decision knowledge in continuous software engineering," in *2nd Workshop on Continuous Software Engineering*, 2017, pp. 7–11.
- [7] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, "Tool support for decision and usage knowledge in continuous software engineering," in *3rd Workshop on Cont. Softw. Eng.*, 2018, pp. 74–77.
- [8] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Softw. Eng.: Guidelines and Examples*. John Wiley & Sons, 2012.
- [9] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for softw. engineering research," in *Guide to Advanced Empirical Softw. Eng.* London: Springer, 2008, ch. 11, pp. 285–311.
- [10] K.-J. Stol and B. Fitzgerald, "The ABC of Software Engineering Research," *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 3, pp. 1–51, 2018.
- [11] J. Saldaña, *The Coding Manual for Qualitative Researchers*, 2nd ed. SAGE Publications, 2009.
- [12] A. S. Freire, A. Meireles, G. Guimarães, M. Perkusich, R. M. da Silva, K. C. Gorgônio, A. Perkusich, and H. O. Almeida, "Investigating gaps on agile improvement solutions and their successful adoption in industry projects - A systematic literature review," in *30th Int. Conf. on Software Engineering and Knowledge Engineering*, 2018, pp. 40–45.
- [13] A. Kleebaum, J. O. Johanssen, B. Paech, R. Alkadhi, and B. Bruegge, "Decision knowledge triggers in continuous software engineering," in *4th Int. Workshop on Rapid Cont. Softw. Eng.*, 2018, pp. 23–26.
- [14] J. O. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech, "Towards the Visualization of Usage and Decision Knowledge in Continuous Software Engineering," in *2017 IEEE Working Conference on Software Visualization*, vol. 1806. IEEE, sep 2017, pp. 104–108.
- [15] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, "Teaching rationale management in agile project courses," in *16. Workshop Softw. Eng. im Unterricht der Hochschulen (SEUH)*, 2019, pp. 125–132.
- [16] C. Miesbauer and R. Weinreich, "Classification of design decisions - an expert survey in practice," in *7th European Conference on Software Architecture (ECSA'13)*, K. Drira, Ed. Springer, 2013, pp. 130–145.
- [17] F. S. Furtado, G. Alexandre, N. G. de Sá Leitão Júnior, I. H. de Farias Junior, and H. P. Moura, "Knowledge management in a software development organization: Identifying tools, processes and benefits," in *29th Int. Conf. on Softw. Eng. and Knowl. Eng.*, 2017, p. 627.

Initial evaluation of the brain activity under different software development situations

Rustam Ikramov, Vladimir Ivanov, Sergey Masyagin, Ruslan Shakirov,
Ilyas Sirazidtinov, Giancarlo Succi, Ananga Thapaliya, Alexander Tormasov, Oydiyoy Zufarova
Innopolis University
Innopolis, Russia
f.last@innopolis.ru

Abstract—The use of biological signals to understand software development has become more popular in the last few years but poses new challenges with respect to the overall experimental settings. In this paper we present such challenges and the approach we took to overcome them. We illustrate our approach by evaluating two programming situations: pair programming and programming with music. The subjects involved in the experimentation are mostly students, however, in the largest case we involved graduate students coming from industry with at least three years of working experience. The results in general support the validity of this approach and encourage to go further in this research line. Moreover, as a byproduct, the analysis of pair programming confirms, from a biological perspective, early hypotheses that pair programming induces higher level of concentration.

Index Terms—Empirical methods, software experimentation

I. INTRODUCTION

Software is the result of the creating activity of software developers. It is clear that improving developers' working conditions might lead to improving the quality of created software the same as productivity of the employer. It is also should be said that working in an area full of distracting agents or activities will decrease the worker's performance. There are lots of different assumptions and myths on how to improve the software development process, but almost no one has an argued proof. Moreover, there is no a general way for evaluating a developer's physical and mental state. As a consequence the state of mind of developers play a major role in the quality and the productivity of the produced software systems. In the recent years, the research arena has become more aware of this fact and new studies have emerge, some of which also directly analyzing biological signals. However, the overall research field in this area is still in its infancy. This paper presents the early results and challenges of using a full EEG to understand the brain activity during coding, in research that started about two years ago [1]. Specifically, we are trying to evaluate empirically how different settings may induce different brainwaves, and from this, understand the mental states of developers in such different settings and thus devise the most suited for a variety of work tasks and conditions. In this prototypical phase, a wide approach has been taken in collecting and analyzing the data, considering "standard" working tasks, in essence preferring breadth over depth in the analysis [27].

Our intention is threefold:

- to perform a preliminary observational evaluation of the areas where phenomena could occur for a followup deeper evaluation;
- to gather a better understanding of the opportunities and the problems arising when collecting and analyzing developers data using EEG, in the hope of facilitating future research;
- to expand our research by supplying our initial results to researchers and research groups interested in replicating our findings.

We have considered two settings, primarily because they represented two situations already present in our working context:

- developing using pair programming (the largest part of the experiment);
- developing with music in the background (still considered, given the interest of the involved researchers).

Notice that we have decided to include in this paper also the small portion of collected data referring to programming with music, as it uses a different experimental protocol which adds a significant breadth at this initial investigation.

The subjects involved in our research belong primarily to two groups:

- graduate students with at least three years of working experience in the industry, who can be assimilated to professionals
- undergraduate students

The unique contribution of this work is that the results in general support the validity of this approach and encourage us to go further in this research line. Moreover, as a byproduct, the analysis of pair programming confirms, from a biological perspective, early hypotheses that pair programming induces higher level of concentration; this appears quite remarkable.

This paper is organized as follows. Section II presents the overall background of the paper. Section III outlines the approach taken to analyze the data and how the data was collected. Section IV details the analysis of the data that we have collected. Section V summarizes the early results that we have obtained so far. Section VI outlines the challenges that we have faced in this kind of empirical work to share it with

other researchers worldwide in the quest of identifying best practices. Section VII draws some conclusions.

TABLE I
SUMMARY OF THE EXPERIMENTS

Id	Situation	Subjects	N	Analysis
1	Pair programming	Graduate students with working experience	10	ERD
2	Pair programming	Undergraduate	3	Correlation
3	Programming with music	Undergraduate	2	ERD
4	Programming with music	Undergraduate	2	Correlation

II. BACKGROUND

As mentioned, there has been an increased interest in using biological signals to understand the mind of developers, in particular using three main kinds of devices:

- electroencephalogram (EEG),
- functional magnetic resonance imaging (fMRI),
- various bio-metric sensors.

a) Electroencephalogram: This is the technique we are considering. To our knowledge, so far a complete portable EEG device has been used in areas related to software engineering only in the study conducted by Lee et al. (2016) [16] on exploring how the mind of developers evolved from novice to experts in program comprehension tasks.

b) Functional magnetic resonance imaging: Functional magnetic resonance imaging (fMRI) provides indirect estimation of brain activity, measuring metabolic changes in blood flow and oxygen consumption as a result of increased underlying neural activity. This technique allows the detection of active regions of the brain [8]. As a result, fMRI is widely used to determine specific brain regions which are responsible for the certain mental activity. In order to learn about software developers' brain activity, researchers chose code review and code comprehension as the primary activities for which brain activity needs to be understood [8], [22], [23].

Siegmund et al. (2014) detected activation specific Broadmann-areas during code comprehension [22]. In their followup work (2017) they investigated the difference between bottom-up program comprehension and comprehension with semantic cues in terms of brain areas involved [23]. This study uses very accurate techniques to explore the work of the brain, the fMRI. Floyd et al. (2017) have performed a similar study applying fMRI to understand the mental activities surrounding program comprehension [9].

c) Ensemble of bio-metric sensors: An alternative approach has been to use an ensemble of bio-metric sensors like eye trackers for measuring pupil size and eye blinks, electroencephalography to determine brain activity, electrodermal activity sensors to detect skin-related activity, and heart-related sensors [10], [17], [29].

TABLE II
TECHNICAL CHARACTERISTICS OF THE MITSAR SMART-BCI EEG DEVICE

Options	Smart BCI EEG headset
EEG channels	24
Poly channel	1 for ECG
Reference	A1, A2, (A1+A2)/2, Cz, REF
Frequency band	0(DC) 70 Hz
Sampling rate	2000 Hz
Storage rate	250 Hz
Noise	1.2 μ V peak-to-peak
Input range	$\pm 300\mu$ V

This approach was applied in a series of investigations which will be described below. The main interest in these investigations was to obtain metrics that correlate with software developers performance. Züger and Fritz (2015) used interruptibility [29] while Müller and Fritz (2015) used positive and negative emotions of software developers [17] as metrics of progress in the change task. They processed the data from multiple bio-sensors and applied methods of supervised learning (Naive Bayes) to distinguish levels of these cognitive states [17], [29].

In these studies, monitoring the state of the mind in depth was limited because:

- the assessment of emotions was performed subjectively by the participants [17];
- a single channel EEG device was used, which may result in an error of up to 50% [21].

III. APPROACH TO DATA ANALYSIS

a) Infrastructure used: We used wireless 24 channel Mitsar SMART-BCI elastic cap for our experiment (details are in Table II). The placement of electrodes was according to the standard 10-20 scheme. Technical characteristics of the Mitsar Smart-BCI EEG device are presented in II. One of the very important steps of EEG recording is the preparation of the EEG cap. We used the canonical type of cleaning before the experiment which is cleaning with spirit. During the data recording, we also used conductive gel to provide a better connection between electrodes and scalp.

Since we use a multi-channel EEG device, the first step to undertake is to select the channels that are the core of the analysis. On the one hand, many channels provide a wide range of information from the whole scalp. On the other hand, this information can be redundant. Moreover, electrodes placed on different parts of the scalp are affected by different types of EEG artifacts, e.g. frontal electrodes are more likely to be affected by muscle and eye movements. During the experimental set up of the device, we found out that a signal from the frontal electrodes cannot be cleaned with EEG preprocessing techniques like Individual Component Analysis and manual filtering. We did not propose any other methods than these two for frontal electrodes since we found out for this particular experiment, central electrodes would

be enough for the analysis and result. Based on this fact we decided to analyze only central electrodes (F3, Fz, F4, C3, Cz, C4, P3, Pz, P4) since they provide proper quality data which can be used in further analysis.

The collected data have a lot of interference including:

- imperfection of EEG equipment;
- metal objects nearby;
- Wi-Fi and mobile network, mobile phones;
- artifacts from the person (e.g. blinking, jaw movements, sneeze);
- harsh background sounds;
- size of the cap.

Moreover, there are patterns to take into account, like the age, the gender, and other physiological characteristics of the subject.

Therefore, after the selection of the channels, we have performed a cleaning of the data with the following filters:

- **Amplitude filtering:** All data which was not in the range $[-200\mu V; +200\mu V]$ we considered as an artifact and removed from the signal. If the total share of noisy data in the channel was more than 20% we considered the channel as compromised and removed it from the dataset.
- **High and low pass filters:** The range of filter was picked according to the possible variance of individual alpha and theta waves and equaled to $[2Hz; 15Hz]$.
- A **notch filter** was used to remove the noise from AC lines.

b) *Processing of the EEG data:* As mentioned above, we decided to use only clean channels (data). The choice of clean channels was reasoned by EEG artifacts that are very hard to be recovered to the original data. Moreover, we use the following infrastructure:

- **Programming tools:** Anaconda 3 Python distribution, NumPy, and SciPy packs, MNE 0.16.1
- **Electrodes:** 'F3-Cz', 'Fz-Cz', 'F4-Cz', 'C3-Cz', 'C4-Cz', 'P3-Cz', 'Pz-Cz', 'P4-Cz' (depending on the setting)
- **Filtering:** Finite impulse response method, as provided by MNE library

Our approach is described in Algorithm 1, implemented, as mentioned using Python 3 with scipy and numpy libraries.

c) *Analysis of the EEG data:* The first step of analysing the data is an adjustment of alpha and theta waves ranges since they could be different for various ages. The variability of alpha waves in age-matched groups has been shown to have a normal distribution ($\mu = 10Hz$, $\sigma = 1Hz$) and exhibits tonic changes, increasing from childhood to adulthood, then declining according to the following formula [13]: $PeakAlphaFrequency = 11.95 - 0.053 \times Age$

We computed peak alpha frequency for each participant (or Individual Alpha Frequency - IAF) and used as the anchor point for calculating alpha sub-bands.

The importance of alpha sub-bands comes from the fact that they improve the accuracy of amplitude measures and more accurately reacts on functional differences of the different oscillators, i.e., functional groupings of neurons, which

Data: EEG measurements of participants

Result: ERD Distributions

for each measurement in Data **do**

IAF(individual α frequency) = $11.95 - 0.053 \cdot Age$;
 theta = [IAF - 6; IAF - 4];
 L1A = [IAF - 4; IAF - 2];
 L2A = [IAF - 2; IAF];
 UA = [IAF; IAF + 2];
 fft = FFT(measurment) erdall= (calibration
 (participant) - fft) / calibration(participant);

end

for each subband **do**

erd [subband] = mean(erdall [subband]);

end

Algorithm 1: ERD distribution calculating algorithm

contribute to alpha power. For instance, the phasic changes in the lower-1 alpha (L1A) and lower-2 alpha (L2A) sub-bands are considered to be as an indicator of task-related attentional demands including both components of attention - alertness and arousal [14]. On the other hand upper alpha (UA) changes correlates with semantic memory processing and synchronization in the theta band reflects episodic memory and the encoding of new information [14]. Concluding all above it can be said that our choice of features depended on the connection between the EEG feature and the cognitive processes that this feature can represent.

In our study we used these ranges of sub-bands:

- L1A range is [IAF - 4Hz ; IAF - 2Hz]
- L2A range is [IAF - 2Hz; IAF]
- UA range is [IAF ; IAF + 2Hz]
- Theta range is [IAF - 6Hz ; IAF - 4Hz]

Next step is counting the number of waves included in the corresponding interval. In this way, we can evaluate the brain activity at each time point.

The analysis is then centered in two main techniques:

- ERD,
- Correlations of brainwaves.

The **ERD** (Event-Related Desynchronization) is a measure of the level to which neurons no longer oscillate in synchrony as they become activated to process the given task [5]. Consequently, more task demanding work should cause bigger ERD difference between rest and programming periods. ERD is calculated as it is shown in the formula below:

$$ERD = \frac{(amplitude)_{rest} - (amplitude)_{programming}}{(amplitude)_{rest}} \times 100\%$$

The ERD is computed for 2000ms window of the signal via Fast Fourier Transformation (FFT). As a result, we obtain a time-series or distribution of ERD for each sub-band for each different programming activity. The name convention of the ERD time-series is presented in Table VI.

Intuitively calculating ERD is subtracting the values of the spectrum from calibration value and normalizing on the

calibration value. As a result we obtain a normalized spectrum of difference in which we find a mean value for the specific frequency ranges. We performed this procedure for each channel and calculated resulted distributions as the average among all channels. For example, we can have active spectrum only for alpha and theta waves as seen from Table III in case of pair programming. This implies that the result can vary and we can get active spectrum for other different waves based on different ERD value based on different types of experiment.

The analysis of the **correlation of brainwaves** identifies the relationships existing among theta and L1-alpha waves, L2-alpha and upper alpha waves. Strong correlations explain different mental activities and statuses.

For instance from all the data obtained from EEG, individual L1-alpha waves stands out as a measure that can be correlated with other brainwaves such as L2-alpha or upper alpha waves. For example, in our studies correlation between these waves in case of pair programming was slightly higher as compared to solo programming whereas this correlation was lower in the case of programming with music rather than without music. These examples from our study imply that correlation can differ affecting the results to be higher or lower depending on the type of experiments we are performing.

d) Experimental protocol: In all cases the students were divided in two groups: treatment and control, even if in one case the control group was very small; again, please remember that the goal of this study is to determine in practice the feasibility of the approach rather than performing sound and reliable observation for the situation under consideration. Each part of the experimentation was scheduled in a separate day and, given the initial availability of two EEG device, when two subjects were involved, they were analyzed together. The following is the detailed steps and here P1 indicates participant one and and P2 indicates participant two.

The steps for the analysis of pair programming have been:

- 1) Calibrating P1 and P2. The calibration part consists of two parts. First one is when subjects sit with closed eyes in front of the computer in a restful state and the second one is the same but with opened eyes. The steps are required to measure alpha and theta synchronizations during calm state.
- 2) Solo programming of P1 and P2 (60 minutes).
- 3) Break, rest period without hard mental activity (10 minutes).
- 4) Pair programming, P1 is on driver mode, P2 is a navigator (60 minutes).
- 5) Break, rest period without hard mental activity (10 minutes).
- 6) Pair programming, P1 is on navigator mode, P1 is a driver (60 minutes).

The steps for the analysis of the effect of music have been:

- 1) Calibration P1 (with and without music): First, the subject sits with the closed eyes in front of the computer in a calm state and for the second time with the same instructions but with opened eyes. As it was mentioned

before, these steps and instructions are necessary to determine the alpha and theta synchronization during the restful state.

- 2) P1 starts programming for the given task without music. (60 minutes)
- 3) Rest period without any types of hard mental activity. P1 is on calm state (break) for 10 minutes.
- 4) P1 starts programming for the given task and listening for a music (the music was chosen by P1 according to his personal preferences). (60 minutes)

e) Description of the collected data: As mentioned, the subjects involved in our research belong primarily to two groups:

- volunteer graduate students with at least three years of working experience in the industry, who can be assimilated to professionals,
- volunteer undergraduate students.

The graduate students were mostly recruited during the so-called “bootcamp,” a two weeks course preparing our students to of preparation to study. Such students are between 23 and 30 years of age and come directly from industry with at least 3 years of experience, so we can consider them almost as professional for the purpose of the generalizability of data.

The undergraduate students were mostly second year students participating at the data collection for curiosity and interest in neurosciences.

Excluding calibration data, the dataset contains 36 hours of recorded EEG data mostly for the analysis of pair programming (11 hours for driver, 11 for navigator and 12 for solo) and 2 hours for programming with music.

IV. ANALYSIS OF THE COLLECTED DATA

Pair programming (PP) is a technique of extreme programming and other agile methods where two developers work together on one workstation, one being the “driver,” who uses the keyboard and write the code, the other being the “navigator” who provides systematic guidance to the driver [12]. Pair programming was picked as a primary topic of the study since it may influence on software developer’s productivity and attention. There have been multiple studies on pair programming evidencing its pros and cons, the pros including: reducing a defect rate, improving the design, increasing productivity [6], and increased concentration of developers [25]. Music Programming is a common practice but, despite of this, rarely investigated: developers and programmers listen to music of their choice while coding.

Our experiment about Pair Programming involved 11 graduate students with ERD to analyse the data and 3 undergraduate students with correlation analysis; our experiment with Music Programming involved 2 undergraduate students and used correlation to analyse the data (Table I).

a) ERD: During the evaluation using ERD, we compare the ERD values in 3 working cases: solo, driver, and navigator. We check the difference between such values using the non-parametric Mann-Whitney test and we determine the significance of the difference. As mention, given the exploratory

goals of this paper we do not systematically track the significance of the result; in this case we use the significance level as an indication of a significant effect of the “treatment,” that is, working in pair or working with music.

Specifically, we consider ERD of theta waves, which desynchronizes (decreases) with the higher memory load, ERD of all alpha ranges (L1A, L2A, UA) synchronizes (increases) with a higher level of attention and semantic memory processing (in other words, the higher value of ERD in alpha band indicates higher attention and semantic memory processing during the given task for the given participant). Using this information we can calculate statistics of ERD distributions of the same sub-bands but from the different activities and compare them.

TABLE III
VALUES OF ERD IN THE FIRST EXPERIMENT

Sub-band	Highest value	Significance	Interpretation
L1A	Pair - navigator	Yes	Higher attention required
L2A	Pair - navigator	No	As above
UA	Not conclusive	No	Nothing
Theta	Solo	No	Usually opposite of L1A, so confirms the results

b) *Correlations*: Using correlations we compare Pearson’s correlation coefficients between the 3 cases of pair/solo programming (solo, pair/driver, and pair/navigator) and the 2 cases of programming with and without music. The brain-waves differ from each other while any kind of mental or physical activity is done by the object. As theta waves decrease with the higher memory load and all the alpha ranges (L1A, L2A, UA) increase with a higher level of attention and semantic memory processing, the correlation of this waves should differ over time. Using this information we can calculate the statistics of the correlation of the same sub-bands from the different activities and compare them. To perform a comparison of Correlation, we performed Pearson’s correlation coefficients (Tables IV and V).

V. RESULTS AND DISCUSSION

a) *Analysis with ERD*: In general, desynchronization in the lower alpha band reflects higher levels of attention [14]; for such band in the case pair programming we obtained the highest ERD for pair-navigator mode and equal values for

TABLE IV
CORRELATION ANALYSIS FOR PAIR PROGRAMMING

Participant	Theta and L1- α	L2-alpha and Upper α
1 (PP-Driver)	0.9	0.8
1 (PP-Navigator)	0.86	0.82
1 (Solo)	0.80	0.86
2 (PP-Driver)	0.799	0.9
2 (PP-Navigator)	0.81	0.85
2 (Solo)	0.84	0.87
3 (PP-Driver)	0.88	0.82
3 (PP-Navigator)	0.93	0.73
3 (Solo)	0.875	0.81

TABLE V
CORRELATION ANALYSIS FOR PROGRAMMING WITH MUSIC

Environment	Theta and L1-alpha	L2-alpha and Upper alpha
With music (Participant 1)	0.825	0.878
Without music (Participant 1)	0.875	0.815
With music (Participant 2)	0.827	0.91
Without music (Participant 2)	0.827	0.835

solo and pair-driver mode (Table III). It may mean that pair programming in navigator mode requires more attention, and this reflects the intuition that the navigator position requires evaluating and guiding the development, which in turn intuitively requires a significant effort of attention, also because the navigator is not involved in a physical contact with the keyboard. The analysis of UA was not conclusive.

According to Klimesch et. al. [14] synchronization in the theta band reflects episodic memory and the encoding of new information. For the theta region we obtained a highest value for solo programming, followed by the navigator, and finally the driver. Theta and alpha waves are supposed to be invariant, which roughly means when alpha increases, theta decreases, and vice versa. As a result, we have that higher desynchronization means lower synchronization. If we denote ERS as event-related synchronization we get the following relation: $ERS_{pair-driver} > ERS_{pair-navigator} > ERS_{solo}$.

Anyway, for now, it is difficult to interpret the meaning of difference in episodic memory working. However, the second part which states the theta band reflects the encoding of new information might be true in case of pair programming.

The analysis of ERD for programming with music did not evidence any specific patterns, perhaps also because of the limited dataset available.

b) *Analysis with correlations*: The analysis of the correlation for pair programming (Table IV) appears somehow to support the claims made with the analysis of ERD. Indeed, the very small dataset is not conclusive for practical reasons, still seeing a second experiment conducted with a different approach hinting at the same pattern as the first one, provides some observational confirmation of the statement that the navigator in pair programming has higher level of attention.

The analysis of the correlation with music (Table V) is again not conclusive, and again we can replicate the limits of the small dataset.

VI. CHALLENGES ENCOUNTERED

Since the goal of this paper is primarily to provide a reference for future experiences in using biological sensors to detect the states of minds of developers, it is important to underline the different challenges that emerged during the experimentations, so that future research can take suitable precautions to mitigate or even eliminate them:

- 1) As this was quite a new experiment in the field of computer science, there was a lack of other works and papers related to the field of computer science to structure our overall experimental setting, therefore it took a considerable effort to define a solid experimental protocol and in due course a significant amount of data got lost.
- 2) The EEG picked up a lot of muscle activity, clouding our data. So subjects had to stay as still as possible and blink as minimum as possible.
- 3) The device could not record from the subjects with the thick hair even with the addition of the gel.
- 4) The EEG experiment was highly influenced by environment noise, so a lot of filtering was done. Location of the experiment highly depends on the goal of the experiment, so it was difficult to find its perfect place.
- 5) Large number of subjects were required and a huge number of experiments were conducted for extraction of useful data and information from the device because the device had poor signal to noise ratio, therefore, this approach is quite effort intensive.
- 6) It took a long time to start the experiment because the device required a complex arrangement of many electrodes around the head with the use of different gels; moreover, also the setup of the computer software required some time.

VII. DISCUSSION AND CONCLUSION

As mentioned, the goal of our work is to provide a new contribution to people interested in performing analysis of software development using biological signals, thus discovering a whole new understanding of the state of mind of developers, who are the main resource in the production of software. To this end we have run four experiments, the largest of which involving 10 graduate students with at least three years of programming experience, so with a professional background similar to developers working in companies, thus providing higher credibility to our observational findings. We have run three additional very small experiments with undergraduate students. The subject of the first largest experiment and of a second small experiment was to analyze pair programming, while the other two small experiments focused on programming with music.

The first result that we have obtained is that, despite several possible challenges, some of which discussed in Section VI, the approach appears to work. For the largest experiment, anyway involving only 10 subjects, we did obtain some observational conclusions confirming previous evidence that pair programming increases the level of attention from a clear biological standpoint. We think that this result is remarkable.

For the case of programming with music, we have not been able to achieve any significant result. We are not discouraged by this – it is an effect of the significant amount of work required to run such experiment and we think that a larger experiment may lead to more conclusive statements.

We have also seen that as the time progresses, indeed, we have become more effective in collecting the required data, so there is an important learning phase that, while it cannot eliminate the significant amount of effort required by this approach, still can partially mitigate it. As a lateral comment, we have not identified any pattern in the data we have lost, so we assume that the results that we have obtained in the largest experiment related to pair programming does not suffer of it.

Moreover a growing number of experiments could be relevant in software relevant for safety critical situations, infrastructures, etc. [2]–[4], [7], [24], [28] or during learning phases [11], [18]. It would also be interesting to involve the open source community in sharing personal data [15], [19], [20], [26].

Summing up, based on all the results, our future work will be based on more focused experimentation on specific programming situations using larger datasets of students and then, indeed, trying to move our analysis to the industry. Also we will try to use not only central electrodes but also the frontal electrodes and for the evaluation, not only correlation and ERD but also other available techniques will be used, thus generating more accurate and comparable results. After applying different approaches for EEG data processing it was found that described correlation methods does not provide veridical outcomes for the further analysis so it should not be used for analyzing EEG data. The observed results might be used for identifying the most productive programming techniques. In the future researches we will test other conditions which may have an impact on developer's productivity.

VIII. ACKNOWLEDGMENTS

We thank Innopolis University for generously supporting this research.

REFERENCES

- [1] S. Busechian, V. Ivanov, A. Rogers, I. Sirazitdinov, G. Succi, A. Tomasov, and J. Yi. Understanding the Impact of Pair Programming on the Minds of Developers. In *Proceedings of the 40th International Conference on Software Engineering Companion, ICSE-NIER'18*, Gothenburg, Sweden, May-June 2018. ACM.
- [2] L. Corral, A. B. Georgiev, A. Sillitti, and G. Succi. A method for characterizing energy consumption in Android smartphones. In *Green and Sustainable Software (GREENS 2013), 2nd International Workshop on*, pages 38–45. IEEE, May 2013.
- [3] L. Corral, A. Sillitti, and G. Succi. Software development processes for mobile systems: Is agile really taking over the business? In *Engineering of Mobile-Enabled Systems (MOBS), 2013 1st International Workshop on the*, pages 19–24, May 2013.
- [4] L. Corral, A. Sillitti, G. Succi, A. Garibbo, and P. Ramella. Evolution of Mobile Software Development from Platform-Specific to Web-Based Multipatform Paradigm. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2011, pages 181–183, New York, NY, USA, 2011. ACM.
- [5] I. Crk, T. Kluthe, and A. Stefic. Understanding programming expertise: An empirical study of phasic brain wave changes. 2015.
- [6] E. di Bella, I. Fronza, N. Phaphoom, A. Sillitti, G. Succi, and J. Vlasenko. Memory processes, brain oscillations and eeg synchronization. *IEEE Transactions on Software Engineering*, 39:930 – 953, 2013.
- [7] E. Di Bella, A. Sillitti, and G. Succi. A multivariate classification of open source developers. *Information Sciences*, 221:72–83, 2013.

- [8] B. Floyd, T. Santander, and W. Weimer. Decoding the representation of code in the brain: an fmri study of code review and expertise. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 175–186, 2017.
- [9] B. Floyd, T. Santander, and W. Weimer. Decoding the representation of code in the brain: An fmri study of code review and expertise. In *Proceedings of the 39th International Conference on Software Engineering, ICSE '17*, pages 175–186, Piscataway, NJ, USA, 2017. IEEE Press.
- [10] T. Fritz and S. C. Müller. Leveraging biometric data to boost software developer productivity. In *Leaders of Tomorrow Symposium: Future of Software Engineering, FOSE@SANER 2016, Osaka, Japan, March 14, 2016*, pages 66–77, 2016.
- [11] I. Fronza, A. Sillitti, and G. Succi. An Interpretation of the Results of the Analysis of Pair Programming During Novices Integration in a Team. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM '09*, pages 225–235. IEEE Computer Society, 2009.
- [12] J. Kivi, D. Haydon, J. Hayes, R. Schneider, and G. Succi. Extreme programming: a university team design experience. In *2000 Canadian Conference on Electrical and Computer Engineering. Conference Proceedings. Navigating to a New Era (Cat. No.00TH8492)*, volume 2, pages 816–820 vol.2, May 2000.
- [13] W. Klimesch. Memory processes, brain oscillations and eeg synchronization. *International Journal of Psychophysiology*, 24:61–100, 1996.
- [14] W. Klimesch. Eeg alpha and theta oscillations reflect cognitive and memory performance: a review and analysis. *Brain research reviews*, 29(2-3):169–195, 1999.
- [15] G. L. Kovács, S. Drozdik, P. Zuliani, and G. Succi. Open Source Software for the Public Administration. In *Proceedings of the 6th International Workshop on Computer Science and Information Technologies*, October 2004.
- [16] S. Lee, A. Matteson, D. Hooshyar, S. Kim, J. Jung, G. Nam, and H. Lim. Comparing programming language comprehension between novice and expert programmers using EEG analysis. In *16th IEEE International Conference on Bioinformatics and Bioengineering, BIBE 2016, Taichung, Taiwan, October 31 - November 2, 2016*, pages 350–355, 2016.
- [17] S. C. Müller and T. Fritz. Stuck and frustrated or in flow and happy: Sensing developers’ emotions and progress. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, pages 688–699, 2015.
- [18] W. Pedrycz, B. Russo, and G. Succi. Knowledge transfer in system modeling and its realization through an optimal allocation of information granularity. *Appl. Soft Comput.*, 12(8):1985–1995, Aug. 2012.
- [19] E. Petrinja, A. Sillitti, and G. Succi. Comparing OpenBRR, QSOS, and OMM assessment models. In *Open Source Software: New Horizons - Proceedings of the 6th International IFIP WG 2.13 Conference on Open Source Systems, OSS 2010*, pages 224–238, Notre Dame, IN, USA, May 2010. Springer, Heidelberg.
- [20] B. Rossi, B. Russo, and G. Succi. Adoption of free/libre open source software in public organizations: factors of impact. *Information Technology & People*, 25(2):156–187, 2012.
- [21] I. M. Rytis Maskeliunas, Robertas Damasevicius and M. Vasiljevas. Consumer-grade eeg devices: are they usable for control tasks? *PeerJ*, 4:1–22, March 2016.
- [22] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann. Understanding understanding source code with functional magnetic resonance imaging. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 378–389, 2014.
- [23] J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, and A. Brechmann. Measuring neural efficiency of program comprehension. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 140–150, 2017.
- [24] A. Sillitti, A. Janes, G. Succi, and T. Vernazza. Measures for mobile users: an architecture. *Journal of Systems Architecture*, 50(7):393–405, 2004.
- [25] A. Sillitti, G. Succi, and J. Vlasenko. Understanding the Impact of Pair Programming on Developers Attention: A Case Study on a Large Industrial Experimentation. In *Proceedings of the 34th International Conference on Software Engineering, ICSE '12*, pages 1094–1101, Piscataway, NJ, USA, June 2012. IEEE Press.
- [26] G. Succi, J. Paulson, and A. Eberlein. Preliminary results from an empirical study on the growth of open source and commercial software products. In *EDSER-3 Workshop*, pages 14–15, 2001.
- [27] A. Valerio, G. Succi, and M. Fenaroli. Domain analysis and framework-based software development. *SIGAPP Appl. Comput. Rev.*, 5(2):4–15, Sept. 1997.
- [28] T. Vernazza, G. Granatella, G. Succi, L. Benedicenti, and M. Mintchev. Defining Metrics for Software Components. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, volume XI, pages 16–23, July 2000.
- [29] M. Züger and T. Fritz. Interruptibility of software developers and its prediction using psycho-physiological sensors. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*, pages 2981–2990, 2015.

Finding conservative schema evolutions by analysing API changes

1st Lynda Ait Oubelli
IRIT/INP-ENSEEIH, ONERA/DTIS
University of Toulouse
Toulouse, France
Lynda.Ait-Oubelli@onera.fr

2nd Yamine Aït Ameur
IRIT/INP-ENSEEIH
University of Toulouse
Toulouse, France
yamine@enseeiht.fr

3rd Judicaël Bedouet
ONERA/DTIS
University of Toulouse
Toulouse, France
Judicael.Bedouet@onera.fr

4th Benoît Chausserie-Laprée
CNES- The French Space Agency
Toulouse, France
Benoit.Chausserie-Lapree@cnes.fr

5th Béatrice Larzul
CNES- The French Space Agency
Toulouse, France
Beatrice.Larzul@cnes.fr

Abstract—Because verification and validation are important activities in model driven engineering (MDE), verifying interfaces preservation is considered an interesting step to understand the evolution of data models by analyzing their interfaces. The interfaces defined on a data model can be used to define model evolution correctness using observational semantics. In this paper, we propose an approach that supports rigorous analysis, verification and validation of behavioral re-factoring. Our work addresses the problem of data model evolution in a formal modelling and verification setting. We focus on data conservation in the specific context of space engineering, where data models may involve thousands of concepts, relationships and each concept has a number of fields or attributes and each relationship has a number of properties.

Index Terms—data models evolution, data model transformation, data models comparison (Bi-simulation), graphs, labelled transition system (LTS)

I. INTRODUCTION

Because project stakeholders require an easy and safe (behaviour-preserving) technique to update model-based applications, several approaches based on formal methods have been proposed [1]–[3]. This work gave rise to several formal comparison approaches [4]–[6]. In this paper, we propose an approach that supports analysis of models behavior preservation after re-factoring. It consists in checking that the APIs (Application Programming Interfaces) of a source data model still hold on the target data model. To address the problem of data model evolution, we have identified three requirements.

Accessibility. Access to model concepts shall be preserved after model refactoring i.e. source model *getters*

or *setters* shall be preserved. Accessibility requirement becomes a path problem in a graph.

Cardinalities. The cardinalities defining the extensions of the relationships between source model concepts (specifying the allowed number or range of instances) shall be preserved after model refactoring.

Knowledge. In order to strengthen concepts evolution, a knowledge base can be associated to the refactoring process to define possible knowledge equivalences or relationships between model concepts. Ontologies are good candidates for such knowledge bases [7].

This paper deals with the *accessibility* requirement. It particularly focuses on the models produced in space engineering.

This paper is organized as follows. Section 2 overviews related work. The proposed approach to handle model evolution and data migration is presented in Section 3. Basic definitions are presented in Section 4. Section 5 summarizes our results, overviews our experiments and positions our approach with respect to the state of the art. Finally, section 6 concludes and provides future work.

II. RELATED WORK

The problem of model refactoring has been addressed by several authors with different perspectives.

Application Programming Interfaces (API). They offer operators to process model concepts by encapsulating modelling details. [8] proposed two categories of application interfaces: external and internal ones. External APIs are designed by library maintainers for clients usage while internal ones are used by the library

code itself. To automatically collect refactoring operations between two APIs versions, [8] uses the *Reffinder* tool, which identifies up to different 52 refactoring types between two API versions. The identified refactoring types are structural, only detectable by mechanical transformations. However, [9] observes that APIs breaking changes are not involved in refactorings. In this case, an application built with an older version of the component API, may fail under a new component API version. When the problem is visible, the application fails to compile or to link. Moreover, it may succeed to compile but its behaviour may be altered [10].

Refactoring. Refactoring-based migration tools are discussed in [11] where the research CatchUp tool is used to update applications. It uses refactorings descriptions to help application developers migrate their applications to a new version. It aims to update applications by recording and playing back the refactorings. Only few refactorings have full records and replay support. According to [10], refactoring at model level is inherently more challenging due to difficulties in assessing the potential impact on structural and behavioral features of the software system.

Data models comparison. Authors in [6] address the problem of user interface (UI) evolution. They focus on the user interface behaviour preservation and study the design process of a user interface resulting from the evolution of a former user interface due to the introduction of new devices and/or new interaction capabilities. Interface behaviors are described by labelled transition systems (LTS) and comparison is handled by bi-simulation of LTS. Furthermore, [4] describe how user interfaces equivalence, with respect to their interaction capabilities and appearance, can be measured. The UI divergences are highlighted, and the possibility of leaving these divergences out of the analysis is provided.

Our previous work [12] proposes an intrusive approach to manage model evolution based on structural differences. It results in a set of evolution operators from source to target models. Models are inspected to identify a set of differences and may produce false positives/false negatives.

In this paper, we propose a non-intrusive approach to handle model evolution. Instead of using a syntactical approach, we rely on API preservation. We consider that a data model evolution is correct if the source data model API is preserved in the target one. The approach is based on path access preservation and graph bi-simulation [13].

III. HANDLING MODEL EVOLUTION AND DATA MIGRATION: OUR APPROACH

In order to handle the semantic data changes involved in the development and exploitation of complex systems in a critical application domain like space engineering, we need to design a rigorous protocol to control the semantic model evolution and data migration.

The approach we propose to compare a source and a target data model relies on 4 steps. Each step manipulates graphs to handle modelling language's semantic evolution. Fig. 1 depicts the defined approach.

- **Step 1. Data models refactoring (interpretation).** It identifies, in each data model, the concepts altered by the evolution. Two input data models will be compared according to these shared identified concepts. According to the latter, both source and target data models are interpreted into a shared model. We use labelled directed graphs (LDG) as ground shared model. Two LDG are produced for source (LDG_s) and target (LDG_t).
- **Step 2. Data models projection.** For each LDG produced from Step 1, a set of labelled transition systems (LTS) with different initial states is produced.
- **Step 3. LTS comparison.** The obtained LTS for both source and target data models are compared using a simulation relationship. Each target LTS shall simulate the corresponding source LTS. When all the source LTS are simulated by the target ones, concept access path preservation is ensured.
- **Step 4. Data conservation.** If step 3 succeeds, source data instances conforming to source and target data models are migrated. The migration procedure is defined depending on the kind of established simulation relation of step 3: strong simulation (source data instances are reused) or weak simulation (source data instances are refactored using the API corresponding to the path identified by the simulation relation).

IV. FORMALISATION OF OUR APPROACH

For the *accessibility* requirement identified in section I, and according to step 1, we define *LDG* as the unified ground model in which data models are transposed.

A. A formal model for checking data model evolution

In the following, C , $attr$ and Bt denote the set of data model concepts (classes, entities, etc.) of attributes and of basic types (Boolean, Integer, etc.).

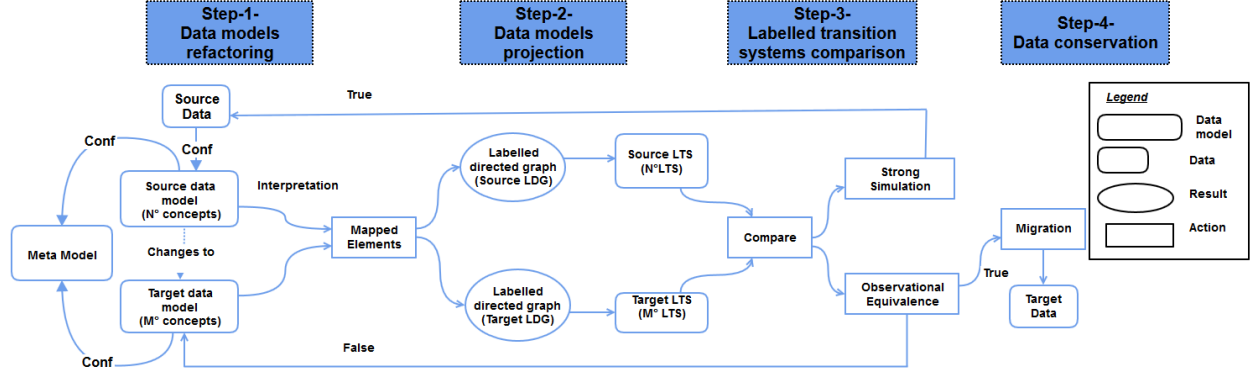


Fig. 1. A four steps based approach for data migration.

Definition 1: A Labelled Directed Graph $ldg \in LDG$ is a graph $ldg = (V, E)$ where

- $V = C \times \mathbb{P}(attr \times Bt)$ is a non-empty set of nodes. Each node represents a concept and its attributes.
- $E \subseteq V \times label \times V$ is a set of directed edges denoting the relations between the concepts.

For any $e = (v_s, l, v_t) \in E$, v_s and v_t represent the source and target node of edge e . Node $v = (c, \{(a_1, t_1), \dots, (a_n, t_n)\}) \in C \times \mathbb{P}(attr \times Bt)$ defines

- c as a concept (class, entity, relation, etc.), with
- $\{(a_1, t_1), \dots, (a_n, t_n)\}$ as a set of typed attributes.

We have considered (l) as a *label* $\subseteq \{isa, refs, haspart, partof, ref, cast, prop\}$ the set of relations for: inheritance *is_a*, aggregation *refs*, composition *haspart*, reflexive composition *partof*, references between concepts *ref*, casting *cast* and association property *prop*. Other relations may be studied for other analyzed data modelling language.

Definition 2: A labelled transition system lts is a structure $lts = (S, s_0, T, \rightarrow)$ where S is a finite number of states, $s_0 \in S$ is an initial state, T denotes a set of labels and $\rightarrow \subseteq S \times T \times S$ is a transition relation. The specific label $\tau \in T$ denotes empty label used to model internal actions, i.e., non observable actions in our approach. We note LTS as the set of lts and T^* as the set of all possible sequences built on labels of T [13].

LTS is the projection of graph LDG on each concept, i.e. each graph ldg has many lts with different initial states corresponding to different concepts.

Step1. Interpretation

Interpretation is the process that produces a graph $g \in LDG$ from a conceptual model $cm \in CM$ where CM is

a set of conceptual models like UML, Entity-Relation (ER), XIF¹.

We denote $CM \xrightarrow{Int} LDG$ and $g = Int(cm)$ the function that describes this process. Each concept (e.g. a class for UML diagrams, an entity for ER, an element for an XIF data model) resp. each concept relation (e.g. inheritance, class association, an entity relation) of cm is interpreted by a node resp. by an edge in the graph g .

Step2. Projection

Projection is the process that produces a set $LTS = \{lts_1, \dots, lts_n\}$ of $lts \subseteq LTS$ from a graph $ldg = (V, E) \in LDG$ where n corresponds to the number of nodes in g . We denote $LDG \xrightarrow{Proj} LTS$ and $LTS = Proj(g)$ the function that describes this process. The following transformation rules for projection define a lts .

Nodes of $V = C \times \mathbb{P}(attr \times Bt)$. For each node $v = (c, \{(a_1, t_1), \dots, (a_n, t_n)\}) \in C \times \mathbb{P}(attr \times Bt)$ in g ,

- the concept $c \in C$ defines a state $c \in S$
- each type $t_i \in Bt$ defines a state $t_i \in S$
- each attribute a_i defines a transition $(c, a_i, t_i) \in \rightarrow$

Edges of $E \subseteq V \times label \times V$. Each edge $e = (v_s, l, v_t) \in E$ where $v_s = (c_s, \{(a_{s1}, t_{s1}), \dots, (a_{sn}, t_{sn})\})$ and $v_t = (c_t, \{(a_{t1}, t_{t1}), \dots, (a_{tn}, t_{tn})\})$ defines a transition $(c_s, l, c_t) \in \rightarrow$.

Initial states for each lts . Finally, each node $v_i \in V$ of the graph $g = (V, E)$ defines the initial state of $lts_i \in \{lts_1, \dots, lts_n\}$.

The projection results in a set of labelled transition systems associated to any data model. Therefore, analysis techniques defined for labelled transition systems can be applied. In particular, our approach uses lts comparison

¹XIF Interchange Format (XIF): A standard in space engineering to define space data models [14].

techniques based on the definition of a simulation relationship.

lts as a model for APIs

An *api* in a set of *API* is made of operations op_i like *getters*, *setters*, *testers* etc. to respectively access, modify or query concepts or attributes of a data model. We note $api = \{op_1, \dots, op_m\} \in API$.

For a given $lts \in LTS$, we say that an $api \in API$ of a given concept c is valid if and only if for each operation $op_i \in api$ there exists a path, starting from the initial state corresponding to the concept c , which accesses each input and output concepts used by any $op_i \in api$. We say that lts satisfies the api API and note $lts \models_a api$.

This definition can be extended to the APIs of any concept in a graph $g = Int(cm)$ resulting from the interpretation of a conceptual model cm . We say that a set $Api \subseteq API$ of APIs defined on g is satisfied if and only if for each $api \in API$ there exists a $lts_i \in Proj(ldg)$ such that $lts_i \models_a api$. We note $g \models_g Api$.

Step 3. LTS comparison

Let g_s and g_t be two *ldg*. Let $lts_s \in Proj(g_s)$ and $lts_t \in Proj(g_t)$ be two *lts* with an initial state associated to the same concept c and api an API defined on the lts_s on the concept c .

We say that api is preserved on lts_t if and only if lts_t simulate lts_s (written as $lts_t \sim lts_s$). Informally, all the paths in lts_s are also paths in lts_t i.e. api is still satisfied in lts_t . Formally, we write

$$lts_t \models_a api \iff lts_t \sim lts_s \wedge lts_s \models_a api$$

Step4. Data conservation

Let g_s and g_t be two *ldg* and Api a set of API defined on g_s such that $g_s \models_g Api$. We say that a set Api of APIs is preserved on g_t if and only if for all $api \in Api$ such that $\exists lts_s \in proj(g_s) \wedge lts_s \models_a api$ there exists a $lts_t \in proj(g_t)$ such that $lts_t \sim lts_s \wedge lts_s \models_a api$. Formally, we write

$$\begin{aligned} g_t \models_g Api & \iff \\ \forall api \in Api, \exists lts_s \in proj(g_s), \exists lts_t \in Proj(g_t). & \\ \text{such that } lts_s \models_a api \wedge lts_t \sim lts_s & \end{aligned}$$

Definition 3: Finally, we say that a **conceptual model cm_t is a correct evolution of a conceptual model cm_s with respect to a set Api of APIs** if and only if

$$g_s = Int(cm_s) \models_g Api \implies g_t = Int(cm_t) \models_g Api$$

Once the conceptual model cm_t is proved to be a correct evolution of cm_s , instances can be migrated. The APIs of the source conceptual model are used to rebuild the instances in the target data model. Some of the produced instances may be partially valued in case cm_t is richer than the source data model.

B. Example

Below, we apply the defined methodology on the example of a conceptual UML class diagram depicted on Figure 2. The objective is to check if the class diagram on the right hand side of Figure 2 is a correct evolution of the one on the left hand side.

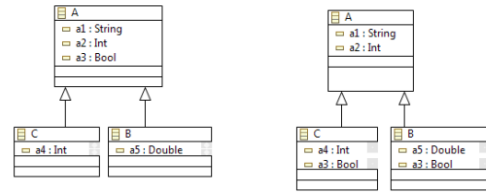


Fig. 2. An example of a data model evolution.

Step1. Interpretation

An example of evolution of an UML class diagram is given in Figure 2. The source and target data models are interpreted using the *Int* function leading to two *ldg*. The source data model contains three concepts A , B and C . Concept B and concept C inherit from concept A . Concept A has three attributes $a1$, $a2$ and $a3$. Concept C has one attribute $a4$ and concept B has one attribute $a5$. In the target data model, we decide to push-down the attribute $a3$ from concept A to both concepts B and C .

Step2. Projection

We project the source and target *ldg* to labelled transition systems. Since three nodes are identified at the *ldg* level, we obtain three *lts* for both source and target *ldg* as shown on Figure 3 for the model on the left hand side of Figure 2. The initial state of each *lts* is one of the three nodes of the associated *ldg*.

Step 3. LTS comparison

In this case study, strong equivalence is not ensured. However, each target *lts* weakly simulates the corresponding source *lts*. One may notice that the opposite does not hold.

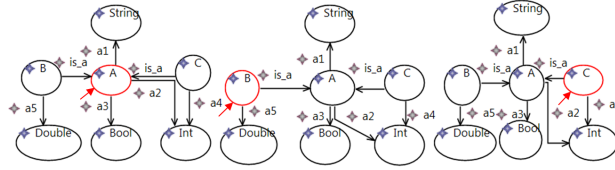


Fig. 3. Projection of a *ldg* to a set of *lts*.

Step4. Data conservation

For data migration, we can assert that the obtained *ldg* and *lts* are conform to Definition 3. The functions of the APIs can be used for data migration.

V. CASE STUDIES

Our approach has been deployed in the space engineering domain. We have studied several case studies with complex data models. In this section, we review the case of the Microscope data model. The Microscope space mission aims at testing the universality of free fall, for the first time in space [15]. In the following, we consider an extract of the data model used to parameterize the telemetry processing and especially to combine two telemetries.

Step1. Interpretation

As shown in Figure 4, it was decided to refactor the data model by replacing two attributes by two composition relationships towards a new class, called *AbstractData*.

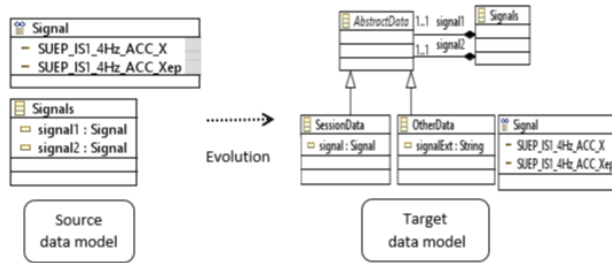


Fig. 4. An extract of a data model evolution in Microscope.

The original attributes *signal1* and *signal2* are factorized into a class *SessionData*, inheriting from *AbstractData* and owning a *signal* attribute of type *Signal*. This way, end-users can combine two telemetries with a known signal or not. Thus, we can identify the following evolutions:

- a new abstract class named *AbstractData* is added;

- two new classes named *SessionData*, *OtherData* inheriting from *AbstractData* are added;
- a new attribute named *signal* of type *Signal* is added to the class *SessionData*;
- a new attribute named *signalExt* of type *String* is added to the class *OtherData*;
- the types of *signal1* and *signal2* are changed from *Signal* to *AbstractData*.

As explained previously, the source and target data models are transformed into two LDG. In the source LDG, the class *Signals* become one concept. In the target LDG, three new concepts appear : *AbstractData*, *SessionData* and *OtherData*.

Step2. Projection

As shown in Figure 5, we project the source and target LDG to one source *lts* *lts_s* and one target *lts* *lts_t*.

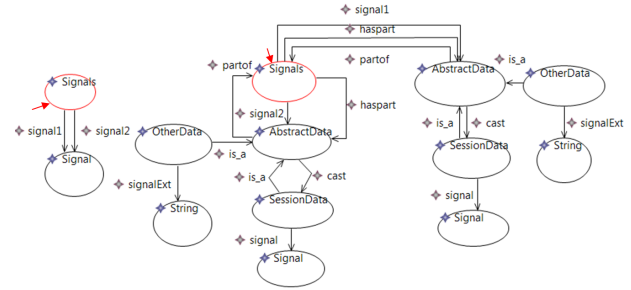


Fig. 5. An extract of the projection of the *ldg* to a set of *lts* in Microscope.

As the *Signals* concept is the only concept that exists in both sides, the initial state of each *lts* is *Signals*.

Step 3. LTS comparison.

As shown in Figure 5, from the initial state *Signals*, all the source transitions are feasible if we cast the target *signal1* and *signal2* to the concept *SessionData*:

$$signal1_{source} = ((SessionData)signal1_{target}).signal$$

$$signal2_{source} = ((SessionData)signal2_{target}).signal$$

Thus, *lts_t* simulates *lts_s* since the following weak simulation binary relationship $R = \{ \langle Signals, Signals \rangle \}$ exists. One may notice that the opposite is false because of the transition *signalExt*.

Step4. Data conservation

During the migration, the values of the old instances of the class *Signals* are preserved, through the creation of two new instances of the class *SessionData*, initialized with these values.

VI. CONCLUSION AND FUTURE WORK

Tool support: The use of the CADP² toolbox helps a lot. Our investigations proved that we have been able to get the diagnosis of the comparison results.

Using template transformations, we were able to transform the Microscope data models to an LDG, then to obtain an internal representation of labelled transition system in AUT (automaton) format. Finally, the simulation between the two automatons is checked, using observational equivalence relationship, thanks to the CADP BISIMULATOR module.

Results and Discussion: In this paper, we presented a semantic observational approach for treating data models evolution. The main interest of the proposed approach is the transposition of the information accessibility in a data model at a logical interface level into a path problem in a labelled directed graph. The approach proved capable to capture all evolutions of a data model into a single logical operator instead of a no-exhaustive list of evolution operators.

Finally, the proposed approach is generic, it is not defined for a single specific data modelling language. It applies to any data modelling language provided that an interpretation of each data model by a *ldg* (from which a set of *lts* is produced) is defined.

Concluding remarks: We believe that addressing the problem of model evolution based on model behavior is promising. Interfaces defined on data models are used to define model evolution correctness using observational semantics. They are also used to prove the existence or the non-existence of composite operators having the property to preserve information contained in original instances.

Relying on labelled transition systems has three potential advantages. First, the overall system is often easier to understand due to the formal and precise nature of the representation scheme. Secondly, the behavior of the system can be analyzed using labelled transition systems theory and associated techniques, which includes tools for analysis. Finally, techniques developed for the comparison of parallel programs can also be applied.

Future work: As a perspective of this approach, we expect to realize a comparative study between the proposed approach and the previous one by comparing traces found by a graph comparison algorithm to structural differences found previously in [12]. We also intend to extend our work to address the evolution of models in presence of cardinalities. Finally, integrating domain knowledge through the introduction of a domain

ontology helps in identifying semantic equivalence at concepts levels and thus address heterogeneous models evolution.

ACKNOWLEDGMENT

Authors would like to express their gratitude to Dr. Raquel Araujo OLIVEIRA for her comments and her constructive suggestions.

REFERENCES

- [1] A. Ferdjouch, A. Baert, E. Bourreau, A. Chateau, R. Coletta, and C. Nebut, "Instantiation of meta-models constrained with ocl: a csp approach," in *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODEL-SWARD)*. IEEE, 2015, pp. 213–222.
- [2] J. E. Rivera, F. Durán, and A. Vallecillo, "Formal specification and analysis of domain specific models using maude," *Simulation*, vol. 85, no. 11-12, pp. 778–792, 2009.
- [3] A. Narayanan and G. Karsai, "Using semantic anchoring to verify behavior preservation in graph transformations," *Electronic Communications of the EASST*, vol. 4, 2006.
- [4] R. Oliveira, S. Dupuy-Chessa, and G. Calvary, "Equivalence checking for comparing user interfaces," in *Proceedings of the 7th ACM SIGCHI Symposium on Engineering interactive Computing Systems*. ACM, 2015, pp. 266–275.
- [5] R. Oliveira and J. Dingel, "Supporting model refinement with equivalence checking in the context of model-driven engineering with uml-rt," in *MODELS (Satellite Events)*, 2017, pp. 307–314.
- [6] A. Chebieb and Y. Ait-Ameur, "A formal model for plastic human computer interfaces," *Frontiers of Computer Science*, vol. 12, no. 2, pp. 351–375, 2018.
- [7] J. Euzenat, P. Shvaiko *et al.*, *Ontology matching*. Springer, 2007, vol. 18.
- [8] R. Khatchadourian and H. Masuhara, "Defaultification refactoring: A tool for automatically converting java methods to default," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 984–989.
- [9] R. G. Kula, A. Ouni, D. M. German, and K. Inoue, "An empirical study on the impact of refactoring activities on evolving client-used apis," *Information and Software Technology*, vol. 93, pp. 186–199, 2018.
- [10] D. Dig and R. Johnson, "How do apis evolve? a story of refactoring," *Journal of software maintenance and evolution: Research and Practice*, vol. 18, no. 2, pp. 83–107, 2006.
- [11] J. Henkel and A. Diwan, "Catchup! capturing and replaying refactorings to support api evolution," in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005*. IEEE, 2005, pp. 274–283.
- [12] L. Ait-Oubelli, Y. Ait-Ameur, J. Bedouet, R. Kervarc, B. Chausserie-Laprée, and B. Larzul, "A scalable model based approach for data model evolution: Application to space missions data models," *Computer Languages, Systems & Structures*, vol. 54, pp. 358–385, 2018.
- [13] R. Milner, *Communication and concurrency*. Prentice hall New York etc., 1989, vol. 84.
- [14] V. Hémery, B. Larzul, and B. Chausserie-Laprée, "Best-ng: a new modeler for describing the satellite's database," in *2018 SpaceOps Conference*, 2018, p. 2305.
- [15] Q. Baghi, G. Métris, J. Bergé, B. Christophe, P. Touboul, and M. Rodrigues, "Gaussian regression and power spectral density estimation with missing data: The microscope space mission as a case study," *Physical Review D*, vol. 93, no. 12, p. 122007, 2016.

²<https://cadp.inria.fr/>

Documenting and Exploiting Software Feature Knowledge through Tags

Marcus Seiler

Barbara Paech

Institute for Computer Science, Heidelberg University, Germany

E-mail: {seiler|paech}@informatik.uni-heidelberg.de

Abstract

Knowledge about features and their relations to detailed requirements or code is important and useful for many software engineering activities such as for performing change impact analysis and tracking feature progress. Documenting feature knowledge is challenging, as companies document features and requirements in issue tracking systems (ITS) and work on code in integrated development environments (IDE). Managing feature knowledge over time is challenging, as features, requirements, and code continuously change. Also, managing the relationships through trace links is challenging, as creating links manually is too time-consuming, and recovering links retrospectively is too error-prone. We developed an approach and tool TAFT to document feature knowledge in ITS and IDE continuously. TAFT uses feature tags to indicate relations between feature descriptions, requirements, work items, and source code. Currently, TAFT comprises a dashboard to track the feature progress, a recommendation system to suggest feature tags for specifications, an inheritor to apply feature tags automatically, and capabilities to navigate in feature knowledge. The tool is integrated into the developers' work environments Jira and Eclipse. In this paper, we present details on the tool support for TAFT, and we report on the results of a case study, which indicates its acceptance.

1. Introduction

Nowadays, many software-developing companies use an issue tracking system (ITS) to support software engineering work [2]. ITS contain several software engineering artifacts like requirements, development tasks (work items), or bug reports. An integrated development environment (IDE) is typically used to work on code that implement artifacts from ITS. Within ITS, requirements are often formulated as requests to modify or to add a specific feature. Since information regarding features is spread across the two sources ITS and IDE, it is hard to document and maintain features

and their relations to code. Explicit knowledge about features and their relations to other artifacts, such as detailed requirements or code is not only useful for software evolution [14], but is also important for many software engineering activities including management tasks such as tracking the feature progress for release planning [9], and including development tasks, such as identifying affected artifacts when performing change impact analysis [11]. While the artifacts of feature knowledge are available in ITS and IDE, in practice their relationship is often managed implicitly or incompletely. This makes it difficult to exploit the feature knowledge for development and management tasks.

In previous work [19], we presented an interview study with practitioners on the use of tagging for feature knowledge and first ideas on a lightweight tagging approach to document feature knowledge. In this approach all artifacts relating to a feature are tagged with the feature. The experts from practice found our ideas beneficial. Documenting feature knowledge using tags seems easy at first glance. However, the experts indicated that providing suitable tags is not that easy and managing feature knowledge across different tools, and over time is a challenging task. The tag consistency over time is challenging, as the development of software systems is characterized by continuous change to features, requirements, and code [7]. In this paper, we present tool support to document, maintain, and exploit feature knowledge with tags in ITS and IDE consistently and efficiently. Currently, our tool support focuses on consistency across different tools and on efficiency to apply tags. The tool support comprises a dashboard to track the feature progress, a recommendation system to suggest feature tags for specifications, an inheritor to apply feature tags automatically, and capabilities to navigate in feature knowledge.

We conducted a case study with students in order to evaluate the acceptance of our approach and tool support according to the technology acceptance model (TAM) [6]. The results show that the students found our approach very useful for navigating to code parts during bug fixing. They rated our approach and tool support as easy to use and emphasized that the approach is intuitive to use. Also, they are motivated to use the approach and tool in future. The

students mentioned the following concerns: cumbersome initial set-up of the dashboard, missing support for viewing metrics from past sprints, and inaccurate recommendations.

The remaining paper is organized as follows: Section 2 introduces the terminology used throughout the paper and our *TAFT* approach. Section 3 describes the details on the tool support. Section 4 describes the case study with its research questions, hypotheses, and results. Section 5 discusses related work. Section 6 finally concludes this paper.

2. Background

This section briefly introduces the terms software features and feature knowledge. Then it describes our approach in more detail.

2.1. Software Features and Feature Knowledge

Various definitions of the term software feature exist in the literature [12, 4, 3]. We adopted the definition by Bosch [4] and define a feature in the context of this paper as a *functional or non-functional property of a software system*.

Different software engineering artifacts that are documented during specification and implementation relate to a feature. A feature description provides a general specification of the feature. Requirements refine the feature. Work items describe development tasks related to realizing the feature. Code implements (parts of) the feature. We therefore define feature knowledge as *knowledge comprising feature descriptions and all related software engineering artifacts such as requirements, work items, and code, as well as their relations*.

Figure 1 shows an example of feature knowledge from the studied project (cf. Section 4). As shown on the left hand side of Figure 1, a feature description was documented during specification and was refined by a requirement afterwards. The work item describes the implementation task for the requirement. Finally, the right hand side of Figure 1 shows the code implementing the functionality of the requirement.

2.2. The Feature Tagging Approach (*TAFT*)

The process of assigning keywords to artifacts is an effective approach to attach additional information to artifacts [20]. We developed a lightweight approach to manage feature knowledge across ITS and IDE [19]. Instead of creating traces between feature knowledge, feature knowledge is tagged with the same keyword. In particular, we use tags for feature descriptions, requirements, work items, and code. The *TAFT* approach works as follows: One tag for each feature of a software is used. The tag summarizes the feature in a short and concise manner. This tagging adheres to the following rules: First, a feature description is tagged with a feature tag if and only if it contains the description

of the feature. Second, a requirement is tagged with a feature tag if and only if the requirement refines the feature. Third, a work item is tagged with a feature tag if and only if the described task addresses specification, quality assurance, or implementation of the feature. Finally, source code is tagged with a feature tag if and only if the source code implements (parts of) the feature.

In the example given in Figure 1, the tag *Transportation* is used to summarize the described feature. The feature tag is applied to the feature description, the requirement, and the work item in Figure 1 as they relate to the feature *Transportation*. The second statement of the code listing in Figure 1 shows the feature tag, as the code implements parts of the feature *Transportation*.

Our approach is independent of the development method used. Thus, *TAFT* is applicable for projects using traditional methods such as waterfall, and for projects using a modern development method such as agile. Moreover, we do not make any assumptions about the cardinality of the relations between features and requirements or between features and code. Thus, it is possible to have requirements, work items, and code tagged with multiple features unlike in the example.

3. Tool Support for *TAFT*

We developed tool support for *TAFT* in Jira¹ and Eclipse², which are common tools for managing software development projects and for working on code, respectively.

Our tool supports the two stakeholders developer and project manager in capturing feature knowledge through labels in Jira and annotations in code. It also supports them in exploiting feature knowledge for development tasks and for management tasks, respectively. We annotate the code with tags instead of tagging commits, as commit messages often contain noise in terms of tangled changes. Tangled changes could result in wrong feature knowledge when tagging a commit [8, 13]. Moreover, code annotations help to understand the code [21] and the cost for creating and maintaining annotations in code is negligible [10]. We rely on Java annotations instead of comment annotations. Unlike comment annotations, Java annotations are language specific, but they retain in the compiled byte code. This is useful for exploiting feature knowledge in (legacy) software even if the source code is not available (anymore).

In the following, we describe the details of the main tool functionality: the feature navigator, the feature dashboard, the feature recommendation and the feature inheritance. The Eclipse plug-in *Feature Navigator* analyzes the source code of a project and scans the code for annotations to support developers. Figure 2b shows a screenshot of the

¹<https://www.atlassian.com/software/jira>

²<https://www.eclipse.org/>

Feature: Integration of an API to retrieve data from public transportation such as stations and their departure schedules.	<pre>// Package, imports and further code omitted @Feature("Transportation") public class OpmvManager implements IOpmvManager { public void queryStation(String stationID) { Request request = new Request.Builder().url(new HttpUrl.Builder() .scheme("http").host("rnv.the-agent-factory.de") .addQueryParameter("stationID", stationID).build()); new OkHttpClient().newCall(request); } }</pre>
Requirement: As a user, I want to click on a station in order to view the departure schedule.	
Work item: Implement service function to retrieve the departure schedule of a station.	

Figure 1: Example of feature knowledge documented in specifications and in code

Feature Navigator. The *Feature Navigator* lists code files implementing a certain feature. The developer can search for features and code files and can directly navigate to a code file once s/he clicked the code in the *Feature Navigator*.

The Jira dashboard *Feature Dashboard* supports project managers in tracking the progress of features in a project. Figure 2a shows the *Feature Dashboard*. The *Feature Dashboard* scans the project for feature tags and displays various metrics, e.g., the number of features, the number of requirements (in this case user stories), and the number of code lines implementing a feature. The metrics are calculated based on the labels applied to issues and the annotations applied to code. The dashboard can be configured for a project and the shown metrics can be selected. Multiple instances of the dashboard are possible to have metrics for multiple projects.

Developers and project managers might not document or update feature knowledge regularly, if the effort is too high. As suggested by Robillard et al. [15], we use recommendation systems to reduce the effort for the tool users. The completion of labels when typing parts of a feature tag is a built-in function of Jira. We extended Eclipse’s code completion capabilities to complete annotations in code when typing parts of an annotation. Both, the label completion in Jira and the annotation completion in Eclipse represent simple recommendation systems. In addition, we developed a Jira plug-in to *recommend feature tags* for issues. Currently, we recommend existing feature tags based on the issue description using a multi-class Naive Bayes classifier. The feature tags are presented to the user together with the confidence score of the classifier. The user is then able to click on the feature tag to apply it to the actual issue. Figure 2c shows the recommended feature tags (Transportation, RouteFinding, Filter, Tweets) with their corresponding confidence scores on the right hand side of the Figure for a user story *Departure Schedule*. In this example, the best matching feature tag is *Transportation* with a confidence score of 74.71% and is shown at the first position of the list. It is up to the user whether to apply one of the recommended feature tags. The classifier is trained iteratively whenever a user applies one of the recommended feature tags. In addition, we provide a feature tag inheri-

tance plug-in for Jira to further reduce the effort to apply feature tags manually. The inheritance plug-in uses existing relations from Jira to automatically apply feature tags for issues in parent-child relations such as a user story consisting of several work items.

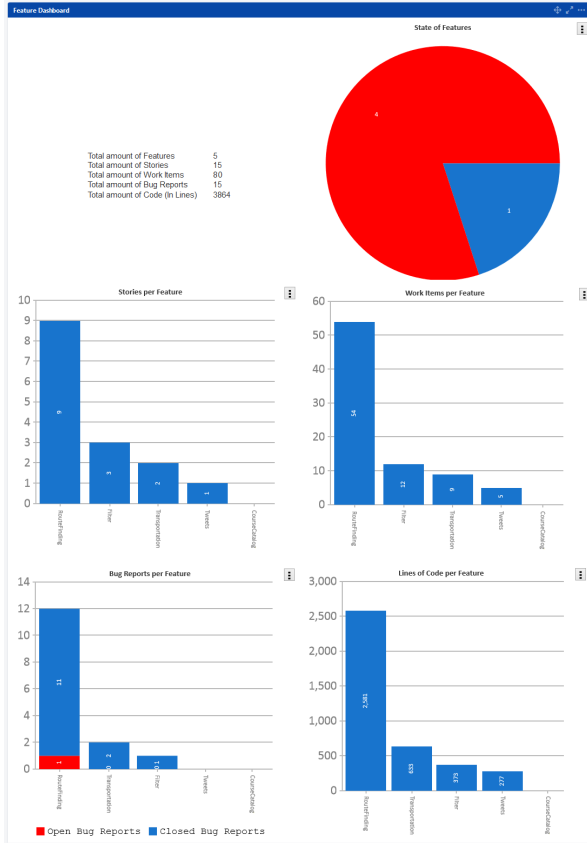
4. Evaluation of TAFT

We conducted a study with students in order to assess the acceptance of the *TAFT*. In the following, we describe the design of the case study and the applied research method in Section 4.1. Section 4.2 presents and discusses the results. Finally, Section 4.3 discusses threats to validity.

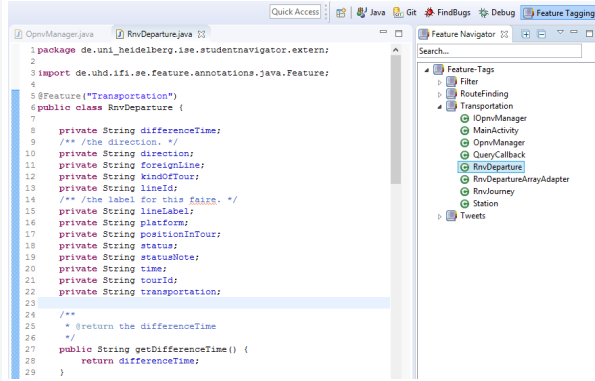
4.1. Case Study Design & Research Method

Study Context: The study was performed during a development project with six students over a period of six months. The project lasted from October 2017 to March 2018. The students developed an indoor navigation app for Android-based devices for a real customer. Primary users are (other) students who use the app to locate and navigate rooms where lectures take place. Also the app is able to retrieve information from public transportation allowing to display the departure schedule of a nearby station. The customer was a mobile development company. The development method was Scrum-like. In each sprint, one of the students acted as Scrum master and thus was responsible for development planning and communicating with the customer. The customer provided a high-level vision description of the app. The students derived features and refined them during development in agreement with the customer. They used Jira with epics to describe features and with user stories to refine features, and with work items to describe development tasks. The students used Git for Java source code and Eclipse as development environment. They applied our *TAFT* approach and used its tool support during the project. At the beginning of the project, the approach and the basic usage of the tool support were introduced. Also, the students were supported in the initial set-up of the tool support.

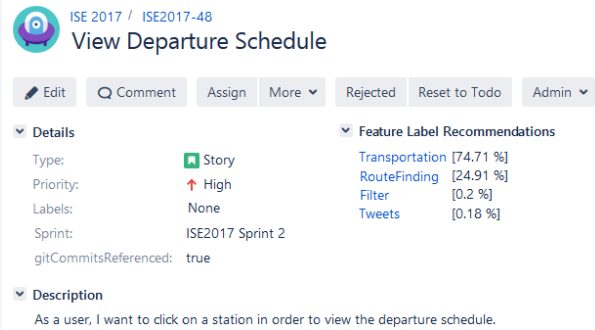
At the end, the project comprised five epics, 17 user stories, 74 work items, and 40 code files. According to our approach, the feature tags were applied to the user stories,



(a) Dashboard to track feature progress



(b) Navigator to find feature in code



(c) Recommendation to suggest feature

Figure 2: Tool support for the TAFT approach

the work items, and the code files. We conducted semi-structured interviews with the students to evaluate the acceptance. The questionnaire used during the interview contained open and closed questions.

Research Questions, Metrics, and Hypotheses: To evaluate the acceptance of our tool, we build upon the Technology Acceptance Model (TAM) by Davis et al. [6], which models the user acceptance of information technology. In our case, the information technology is the approach and the tool support. TAM uses the variables *perceived ease of use*, a *subjects' intention to use* and *perceived usefulness*. We raise the following research questions for acceptance evaluation:

RQ_1 How easy is it to use the approach and tool support?

RQ_2 How useful is the approach and tool support?

RQ_3 Do the students intend to use the approach and tool support in future?

Regarding the tool support, we are particularly interested in the recommendation and the dashboard. For usefulness, we are particularly interested in progress tracking and feature knowledge relations. We provided a questionnaire to

the students with questions corresponding to the research questions.

According to Davis et al. [6] point scales such as Likert scales can be used to measure the variables of TAM. We used a Likert scale with five scale points for asking the students to assess the approach and tool support. The answers to the Likert scale were mapped to an integer as follows: strongly disagree = 1, disagree = 2, neutral (neither agree nor disagree) = 3, agree = 4, and strongly agree = 5.

We use the students' assessments together with their rationale for the ratings as metric for RQ_1 , RQ_2 , and RQ_3 , respectively. Our hypothesis for acceptance is: We expect that the values for the TAM-variables are higher or equal to 3.5. Thus, we expect that most of the responses are in the range between neutral (with a slight tendency to agree) and strongly agree.

4.2. Results & Discussion

In the following, we use the answers to the open questions to provide the details for the assessments of the students.

The students stated that the approach is easy to use (RQ_1) as it is very intuitive to use. All students found it easy to apply feature tags to issues. A minority stated that applying feature tags in code is more complicated. The students needed more effort to equip some code files with feature tags, as these code files implemented multiple features. This is backed up by the numbers of the project. Each of the 17 issues had exactly one of the tags applied. Of the 40 code files, 37 had at least one of the tags applied. The majority (33 code files) contained one tag. One code file each contained two and three tags. Two code files contained four tags. Altogether, it seems easier to tag specifications instead of code files. The recommendation is rated easy to use and a little less useful compared to the usefulness to relate and track feature knowledge. Overall, the students stated that the recommendation works well and that the recommendation can help to prevent incorrect feature tags. However, the students did not use it very often (only for 14.04% of all issues). A minority reported that wrong recommendations for feature tags decrease the usefulness. One reason could be that the students were presented with all feature tags including those with low confidence values. Therefore, we need to study how recommendation usage can be improved.

Overall, the students found the approach useful (RQ_2). The use of the dashboard to track the progress was rated as easy and very useful. The students stated that the dashboard exactly provides the data needed to perform the tracking. However, they missed the functionality to view metrics for past sprints in the dashboard. Also, a minority perceived the initial set-up of the dashboard somewhat cumbersome.

The result for the intention (RQ_3) is rather poor compared to the other two variables. The rather low intention to use the dashboard could be due to the fact that this is mainly helpful for the project manager and the students do not see themselves as project managers in the short future. The students justified the assessment for the intention with the dependency on the project size. In smaller projects with a similar scope as this evaluation project, the students would rather not use the approach and the tools, as the application and maintenance of the tags creates overhead for developers and they can remember the feature knowledge by themselves. In very large projects, students indicated that there could be many feature tags that need to be managed. Therefore, they would apply the approach and tool support in those projects. Some students used the feature tags in code to locate code parts that might be affected by bugs. We plan to investigate whether improved support for locating code affected by bugs would raise the motivation for future usage.

Overall, our hypotheses hold and we conclude that our approach is feasible and accepted. We applied our approach in a new project, but it could be also applied to an existing project. The requirements and the code files of an ex-

isting project must be equipped with feature tags in retrospect to make our approach work. The effort for this re-documentation is considerable as the relations between all requirements and all code files have to be mapped to the features. However, it can be incrementally done during refactoring or other changes to features.

4.3. Threats to Validity

We discuss threats according to Runeson et al. [16].

Construct Validity: The construct validity was ensured through data source triangulation by using direct methods (semi-structured interviews with open and closed questionnaires) as well as indirect methods (review of data produced by tool logging). A possible threat is that researcher and interviewee might interpret the questions differently. The threat is mitigated by the format of face-to-face interviews, which enabled the interviewees to ask questions. Also, another researcher checked the questionnaire for applicability and understandability.

Internal Validity: The students knew that the researcher had developed the approach and its tool support. The threat was mitigated as the researcher appreciated both positive and negative feedback from the students. The motivation of the students to use the approach and its tool support might be influenced by worries about grades. However, the researcher was not involved in the final grading and the usage had no influence on the grades.

External Validity: The documented feature knowledge is specific to this development project and the size of the development project is limited. Moreover, we applied our approach for a new project only. Thus, the results for other projects can be different from the results reported in this study, and the findings cannot be generalized for developers working in industry. However, the project contained situations common to projects in industry, e.g., the elicitation of requirements by the participants, changing requirements due to changed customer needs, as well as communication problems across developers regarding their tasks.

Reliability: One researcher did the interviews and assessments to ensure consistency. Other researchers might interpret the results in another direction. The researcher documented the steps during design, data collection, and analysis. In addition, another researcher reviewed the design of the case study and the steps for analysis to increase reliability. This also ensures the reproducibility of the study.

5. Related Work

Our approach relates to traceability. *TAFT* provides a coarser-grained traceability as commonly used trace links, as the relations between requirements and code files are established on feature level. In [18], we compared the trace links resulting from tags with other approaches to create traceability links.

There are few approaches which also use tags to document and exploit feature knowledge. Mainly they are from the area of product lines where features are used to manage variability. Thus, they are more heavyweight than our approach. Savage et al. [17] present an Eclipse based tool for locating and tracing feature in code. The underlying approach is different from ours, as they do not directly tag code with its implementing features. Instead, a user manually relates code to feature using an annotation after feature location was performed. Similar to the metrics in our dashboard, they provide a view showing the distribution of features across the code. Ji et al. [10] present an approach to equip code with feature annotations. They simulated the development of a product line of cloned projects using the annotation approach. They found that maintaining such annotations in code is not costly, but useful for maintenance tasks. Andam et al. [1] and Burger et al. [5] both present a standalone tool for locating features in code. Both use comment annotations to document features in code and they use feature location techniques to annotate the code (semi-)automatically. Andam et al. [1] also provide a dashboard for viewing feature metrics. In contrast to ours, their dashboard targets at developers by providing more specialized metrics, e.g. nesting depths of annotations. As the tools focus on feature location, all of them provide capabilities to navigate in feature knowledge documented in code. None of the tools tag specifications nor do they provide recommendations or inheritance of tags.

6. Conclusion & Future Work

In this paper, we reported on the tool support of our *TAFT* approach and its acceptance in a case study with students. Overall, the results show that the approach and tool support are accepted.

In future work we would like to address the problems experienced by the students. To further ease the application of feature tags, we are also working on recommendations to suggest feature tags in code and on inheriting feature tags for code. In addition, we want to investigate how practitioners think about our tool, e.g., by performing interviews with practitioners.

Acknowledgement. We would like to thank all students for their effort in this study.

References

- [1] B. Andam, A. Burger, T. Berger, and M. R. V. Chaudron. Florida: Feature location dashboard for extracting and visualizing feature traces. In *11th Int. Work. on Variability Modelling of Software-intensive Systems*, pages 100–107. ACM, 2017.
- [2] O. Baysal, R. Holmes, and M. W. Godfrey. Situational Awareness: Personalizing Issue Tracking Systems. In *35th Int. Conf. on Software Engineering*, pages 1185–1188. IEEE, 2013.
- [3] T. Berger, D. Lettner, J. Rubin, P. Grünbacher, A. Silva, M. Becker, M. Chechik, and K. Czarnecki. What is a feature?: A qualitative study of features in industrial software product lines. In *19th Int. Conf. on Software Product Line*, pages 16–25. ACM, 2015.
- [4] J. Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press Books. Addison-Wesley, 2000.
- [5] A. Burger and S. Grüner. Finalist2: Feature identification, localization, and tracing tool. In *25th Int. Conf. on Software Analysis, Evolution and Reengineering*, pages 532–537. IEEE, 2018.
- [6] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw. User acceptance of computer technology: A comparison of two theoretical models. *Manage. Sci.*, 35(8), Aug. 1989.
- [7] M. W. Godfrey and D. M. German. The Past, Present, and Future of Software Evolution. In *Frontiers of Software Maintenance*, pages 129–138. IEEE, 2008.
- [8] K. Herzig and A. Zeller. The impact of tangled code changes. In *10th Work. Conf. on Mining Software Repositories*, pages 121–130. IEEE, 2013.
- [9] S. Jantunen, L. Lehtola, D. C. Gause, U. R. Dumdum, and R. J. Barnes. The Challenge of Release Planning. In *Int. Work. on Software Product Management*, pages 36–45. IEEE, 2011.
- [10] W. Ji, T. Berger, M. Antkiewicz, and K. Czarnecki. Maintaining feature traceability with embedded annotations. In *19th Int. Conf. on Software Product Line*, pages 61–70. ACM, 2015.
- [11] N. Kama. Change Impact Analysis for the Software Development Phase : State-of-the-art. *Journal of Software Engineering and Its Applications*, 7:235–244, 2013.
- [12] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.
- [13] H. Kirinuki, Y. Higo, K. Hotta, and S. Kusumoto. Hey! are you committing tangled changes? In *22nd Int. Conf. on Program Comprehension*, pages 262–265. ACM, 2014.
- [14] L. Passos, K. Czarnecki, S. Apel, A. Wąsowski, C. Kästner, and J. Guo. Feature-oriented software evolution. In *7th Int. Work. on Variability Modelling of Software-intensive Systems*. ACM, 2013.
- [15] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann. *Recommendation Systems in Software Engineering*. Springer, 2014.
- [16] P. Runeson, M. Host, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, Hoboken, NJ, USA, 1st edition, 2012.
- [17] T. Savage, M. Revelle, and D. Poshyvanik. Flat3: feature location and textual tracing tool. In *32nd Int. Conf. on Software Engineering*, pages 255–258. IEEE, 2010.
- [18] M. Seiler, P. Hübner, and B. Paech. Comparing traceability through information retrieval, commits, interaction logs, and tags. In *10th Int. Work. on Software and Systems Traceability*. IEEE, 2019. (accepted to be appear).
- [19] M. Seiler and B. Paech. Using tags to support feature management across issue tracking systems and version control systems. In *23rd Int. Work. Conf. Requirements Engineering Foundation for Software Quality*, pages 174–180. Springer, 2017.
- [20] M.-A. Storey, J. Ryall, J. Singer, D. Myers, L.-T. Cheng, and M. Muller. How Software Developers Use Tagging to Support Reminding and Refinding. *IEEE Transactions on Software Engineering*, 35:470–483, 2009.
- [21] M. Sulír, M. Nosál', and J. Porubán. Recording concerns in source code using annotations. *Computer Languages, Systems & Structures*, 46:44–65, nov 2016.

Research on Scheduling Area Partition Method Based on Multiple Algorithms

Lefeng Li

College of Computer Science
Inner Mongolia University
Hohhot 010012, China
E-mail: 932164210@qq.com

Shanshan Wang

College of Computer Science
Inner Mongolia University
Hohhot 010012, China

Corresponding author e-mail: cswangss@imu.edu.cn

Abstract—Shared bicycle, as a green means of transportation, is very popular among people and it is an important way for many people to travel daily. In recent years, with the increasing scale and frequency of bike sharing system, the unbalanced use of shared bicycle has a great impact on the users' experience, which is one of the main problems faced by current system operators. Division of the traffic area can not only provide a new idea for solving the problem of unbalanced bike usage, but also provide a theoretical and practical basis for the planning, layout, construction, operation and scientific dispatching of shared bicycle system. However, there are few clear methods to study the partition method of shared dispatching area. To solve this problem, based on historical bicycle data, traffic station data, we analyze the rules of shared bicycle space-time characteristics and propose a method of dividing shared bicycle dispatching areas by combining K-medoids clustering, association rules and total demand constraint adjustment. We evaluate our approach on the New York City (NYC Citi Bike) bicycle sharing system and show the advantages of our approach for Large-scale station-level dispatching area optimization (beyond baseline approaches).

Keywords—scheduling area partition method; bike sharing system; rebalance; clustering; Total demand optimization

I. INTRODUCTION

Bicycle sharing system is widely used domestically and abroad. It provides great help to solve the problem of "last kilometer" and traffic jam. A user can rent (i.e. check-out) a bike at a station near their origin and return (i.e. check-in) it to a station close to their destination. A record is generated when a bicycle is borrowed/returned, including the location of the origin station, the location of the destination station, and the duration of the ride.

However, bicycle sharing system still faces challenges in bike rebalancing between stations. Essentially, bike usage is constrained by time and location, so the traffic in the whole city will be unbalanced. For example, some stations may have a lot of bicycles returning, but some stations may not have available bicycles for users. In order to solve this problem, Most studies directly predict bicycle demand during a future period in order to avoid unbalanced problems and dispatchers manually schedule ahead of time, but the accuracy is not high, and when the number of stations is large, dispatchers need to schedule each station manually, which increases the workload and difficulty of delivery personnel and makes it difficult to

achieve efficient and orderly scheduling. Thus, in order to reduce the workload and the difficulty of their jobs, therefore makes it difficult to achieve efficient and orderly scheduling. Thus, in order to reduce the workload and difficulty, we start with reducing the scale of the problem, which is to divide the whole traffic into different areas. The location, station-station trip frequencies as self-fluidity and the bike demands are taken into account, and the areas are adjusted by constraints to achieve the optimal results of the maximum balance within the areas. This not only reduces the workload and difficulty of dispatchers, but also experiments (see Chapter IV, Part C) show that the demand distribution of bike usage of cluster is more regular than that of station-level bike usage. Therefore, our proposed method lays a foundation for improving the accuracy of bike usage demand prediction in the future.

In these days, the scanning method and the clustering algorithm [1] are mainly used domestically and abroad to divide the vehicle scheduling area. Among them, the scanning method [2] is only applicable to the cases where the number of users is small and the distribution area is not large. Although the spatial clustering algorithm [3] applicable to the cases of a large number of customers and a large scheduling area, there is still a lack of criteria for reasonably allocating the weights of spatial and non-spatial attributes, or a lack of accurate methods to define the distance between feature attributes. If applied directly to dynamic scheduling of shared bicycle system, it may lead to large random errors.

Bike sharing system has a large number of stations and complex attributes, which makes it difficult to calibrate their attributes one by one. However, due to the self-fluidity of shared bicycle [4,5], there is a strong correlation between some stations. Therefore, if the association rules are used to collect stations with strong correlation and adopt K-medoids algorithm [6,7] and constraint adjustment for scheduling, it can effectively avoid large random errors.

Based on the above analysis, considering the actual bike demand of real-time dispatching of bike sharing system, we propose a scientific method for dividing the shared bicycle scheduling area by combining K-medoids clustering, association rules and restraint adjustment of total demand in the bike sharing system. In the first step of this method, K-medoids clustering is applied to the stations of bike sharing system. Then on the basis of the self-fluidity between stations and the transformation relationship between stations, the set of strong

association rules is screened out by using association rules. According to the constraints of the total demand in the areas, the total demand within the areas is optimized, and the regional division of dynamic dispatch of urban shared bicycle system is finally realized.

II. DESIGN OVERVIEW

We provide an overview of the symbols used in this paper (Table I) and problem definitions, as well as the description of design approach.

A. Basic Definitions

Definition 1: Station information. A station $S_i = (id, lon_i, lat_i)$ denotes station information, where id represents the unique identity of each station, lon_i is the longitude of the station, lat_i is the latitude of the station.

Definition 2: Trip. A Trip $T_r = (s_o, s_d, \tau_o, \tau_d)$ is a historical bike usage record, where s_o denotes the origin station, s_d is the destination station; τ_o and τ_d are the time when bike is checked out at s_o and checked in at s_d , respectively.

Definition 3: Demand of bicycles. In time t , given a set of Check-out of station S_i , $O_{S_i}(t) = \{O_{S_1}, O_{S_2}, \dots, O_{S_n}\}$ and check-in of station S_i , $I_{S_i}(t) = \{I_{S_1}, I_{S_2}, \dots, I_{S_n}\}$. We want to get the demand of each station $S_i.d(t) = I_{S_i}(t) - O_{S_i}(t)$.

Problem Definition: Scientific division of regions. Given a set of stations $S_i = \{S_1, S_2, \dots, S_n\}$, we want to cluster each station S_i to form $C_{1, i} = \{C_{1,1}, C_{1,2}, \dots, C_{1, k}\}$ clusters.

TABLE I. NOTATIONS

Notation	Description
N	Number of historical bike usage records
ρ	Coefficient of normal bike transaction records
n_b	Number of shared bikes
t	Days of data acquisition
λ	Coefficient of bike flow distribution
ω	Coefficient of on-frame mobile bike
$r_{i,j}$	Correlation coefficient from station i to station j
$n_{i,j}$	Number of bikes from station i to station j
n_i	Number of bikes flowing out from station i
S_i	The i^{th} station
T_r	A bike usage record
C_i	The i^{th} cluster
$O_{S_i}(t)$	Check-out of station S_i in time t
$I_{S_i}(t)$	Check-in of station S_i in time t

B. Design Methodology

Despite the time and location of the user's choice of borrowing is random, bikes are bound to be checked in at some station. Based on this simple observation, bike sharing system is decoupled in Figure 1 into two parts by analyzing the

mobility [8,9] of bikes and characteristics. Based on historical bike usage records, we first use statistical methods to analyze the spatial and temporal distribution characteristics of bikes. Then, considering the space-time distribution characteristics of shared bikes, we propose a scientific method for dividing dispatching areas (see chapter III). Finally, the experimental results of NYC Citi Bike System show the advantages of our method.

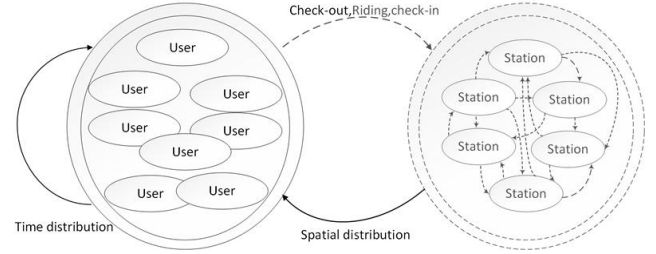


Figure 1. Components of a bike-sharing system

III. SCHEDULING AREA PARTITION ALGORITHMS

In this chapter, firstly, the spatial and temporal distribution characteristics of bike sharing system operation data are analyzed. The purpose is to grasp the operating regularity of the system, mine the trip patterns, obtain the macro operation rules of each station, and improve the quality of the dynamic scheduling of the bike sharing system. Then we put forward a scientific division method and show the specific steps of implementation through the above analysis. The goal of scientific partitioning is to transform the problem from a complex one (about 1,000 predictions per hour) to a simpler one, thereby reducing the complexity of the problem, making it easier to handle and helping to avoid over-fitting.

A. Spatiotemporal Analysis

Distribution Characteristics. As shown in Figure 2, from the macro-analysis [10] of the impact of month on the shared bicycle demand, it can be seen that there is a regular pattern of increasing demand from April to June. From June to September, the demand is stable but still in a high level. Selecting these months is conducive to dealing with the imbalance during the peak period. The figure on the right specifically shows the time distribution regular pattern of two different parameters, stations and time. It can be found that the early peak appears around 8 o'clock and the late peak appears around 5 o'clock.

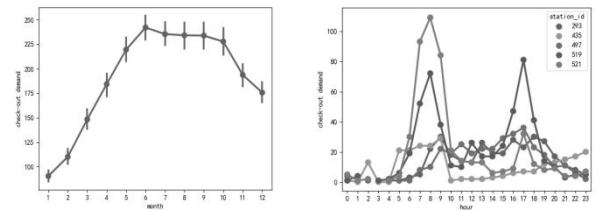


Figure 2. Law of time distribution

Analysis of Spatial Distribution Characteristics. Spatial distribution [11] characteristics analysis is based on each station as the research object. It analyses the distribution regularity of the whole city and the stations to which the vehicles borrowed from the station are returned or the vehicles

returned to the station are borrowed from the station, and calculates the correlation between the stations. Station correlation refers to the flow correlation between two stations. We use the correlation coefficient to express their correlation. The larger the value, the more frequent the bicycle flows between the rental points, the greater the travel demand of users in this area. The formula can be expressed as follows.

$$r_{i,j} = n_{i,j} * n_i^{-1} \quad (1)$$

Through the analysis of the spatial distribution characteristics of the stations, the borrow-return flow relationship between the stations in the system is determined, which provides the data basis for clustering the dispatching areas.

B. Specific realization

Figure 3 presents the iterative procedure of the partitioning method which organically combines three factors (location, self-fluidity and bike usage demand) of the stations. Stations within the same circle represent a cluster. The algorithm repeats the following three steps in each iteration: Geo-clustering, Strong Association Rule generation and constraint adjustment.

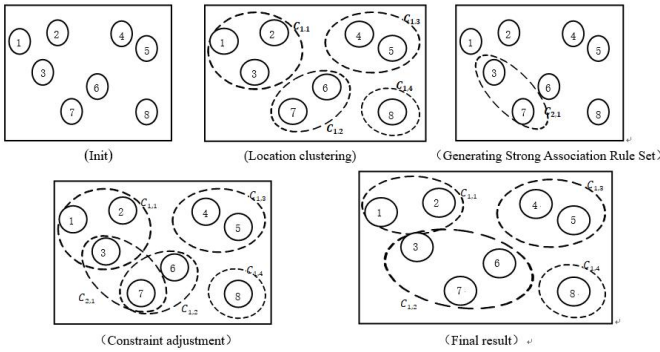


Figure 3. Partitioning method procedure

- **Location clustering.** According to the geographic location of each station, it is clustered into $K_1 = \{C_{1,k}\}_{k=1}^{k_1}$ by K-medoids method, this is the first time that location clustering is performed on all stations in the shared system”.
- **Strong Association Rule Set generation.** check-in/check-out between each station is calculated by the statistics of historical bike usage records, and the strong association rule set $\{C_{2,k}\}_{k=1}^{k_2}$ is screened by Aprior algorithm [12,13]. In this paper, we select 7:00-9:00 in the early peak period and 17:00-19:00 in the late peak period as the basis for screening strong association rule sets. In the process of preliminary classification of shared bicycle stations by association rules, the minimum support threshold Sup_{min} is very important, which determines the quality of clustering in the next step. If the value is too small, the correlation between the stations in the set is very weak, it will bring great errors to the later clustering to divide the

dispatching area. If the value is too large, some stations with correlation may be screened out. When the next clustering division is carried out, most stations with less correlation will be introduced, which will also lead to larger errors in the result of division. Since there is no general method to determine the minimum support threshold, which is usually set artificially according to specific conditions, this paper considers that in a bike sharing system, the minimum support threshold should be determined according to various factors, and the expression is as follows.

$$Sup_{min} = \frac{N * p}{t * \omega * n_b} * \lambda \quad (2)$$

A frequent itemset [14] is formed by selecting the records whose correlation coefficients are greater than those of the bike usage records. The relevant stations are put into the same set by using association rule algorithm. The stations in the set are the result of the users' free choice of the place to rent or return the bike when they travel. The principle of dividing the dispatching area is to excavate the travel rules of the users and balance the task of dispatching the vehicles. To improve scheduling efficiency, the correlation set meets the precondition of adjusting clustering in the next step.

- **Constraint adjustment.** The cluster $\{C_{2,k}\}_{k=1}^{k_2}$ obtained in step 2 and cluster $\{C_{1,k}\}_{k=1}^{k_1}$ obtained in step 1 are calculated as follows.

$$\{C_{2,k}\}_{k=1}^{k_2} \cap \{C_{1,k}\}_{k=1}^{k_1} \quad (3)$$

where $\{C_{1,k}\}_{k=1}^{k_1}, \{C_{2,k}\}_{k=1}^{k_2} \neq \emptyset$. If the result calculated by (3) is $C_i \{C_{1,k}\}_{k=1}^{k_1}$, it is the result of optimization, On the contrary, s_i in C_i is calculated in $\{C_{1,k}\}_{k=1}^{k_1}$ and $\{C_{2,k}\}_{k=1}^{k_2}$ as follows.

$$C_s(t) = \min \left| \sum_{i=1}^k s_i \cdot d(t) \right| \quad (4)$$

where the $C_s(t)$ value at $\{C_{1,k}\}_{k=1}^{k_1}$ is the smallest, then s_i is classified as $\{C_{1,k}\}_{k=1}^{k_1}$, and vice versa.

Until all the collection in the k_1 clusters: $\{C_{2,k}\}_{k=1}^{k_2}$ are processed.

C. Algorithm Complexity Analysis

Time complexity [15,16]. The time complexity of our proposed method is mainly composed of k-medoids clustering and searching for the minimum total demand. In k-medoids algorithm, each point needs to be enumerated and the sum of its distances to all other points is obtained, so the complexity is $O(n^2)$. In addition, the time complexity of seeking the

minimum total demand is $O(n^2)$. To sum up, the time complexity of the whole algorithm is $O(n^2 + n^2)$, i.e. $O(2n^2)$.

Spatial complexity[17]. The main memory overhead of the algorithm is the calculation of the cluster center and the total demand. The memory overhead can be effectively reduced by calculating the distance of a single data object at a time and the local demand in the morning and evening peak periods, which results in a spatial complexity of $O(n)$, so the spatial complexity of the whole algorithm is $O(n)$.

Algorithm 1: Partition Method Algorithm

Input: Station $\{S_i\}_{i=1}^n$, Trips $\{T_r\}_{r=1}^n$, iteration threshold K, parameters $K_1 > K_2$

Output: K_1 clusters $C_{1,1}, C_{1,2}, \dots, C_{1,k_1}$

```

1: Cluster  $\{S_i\}_{i=1}^n$  into  $K_1$  Clusters,  $C_{1,1}, C_{1,2}, \dots, C_{1,k_1}$  By K-medoids Based on locations;
2: Stations  $\{S_i\}_{i=1}^n$  into  $K_2$  Clusters,  $C_{2,1}, C_{2,2}, \dots, C_{2,k_2}$  By Aprior Algorithm;
3: Initialize  $k=1$ ;
4: While  $k < K$  Do
5:   for  $a=1:n$  Do
6:     for  $i=1:n$  Do
7:       If  $C_i = C_{2,a}\{S_i\}_{i=1}^n \cap C_{1,i}\{S_i\}_{i=1}^n \neq \emptyset$  and  $C_i \neq C_{2,a}\{S_i\}_{i=1}^n \neq C_{1,i}\{S_i\}_{i=1}^n$  Then
8:         Calculate  $C_i(t)$  in the set of  $C_{2,a}\{S_i\}_{i=1}^n$  and  $C_{1,i}\{S_i\}_{i=1}^n$  respectively for S in  $C_i$ ;
9:         If  $S_i \in C_{1,i}\{S_i\}_{i=1}^n$ ,  $\min(C_i(t))$  Then
10:           $S_i$  is classified as cluster  $C_{1,i}\{S_i\}_{i=1}^n$ ;
11:         Else  $S_i$  is classified as cluster  $C_{2,a}\{S_i\}_{i=1}^n$ ;
12:        $k=k+1$ ;
13: Return  $K_1$  clusters  $C_{1,1}, C_{1,2}, \dots, C_{1,k_1}$ 

```

Figure 4. An Algorithm of Partition Method

IV. EXPERIMENTS

In this section, we use our proposed method to construct a model for partitioning the scheduling area, and test our method on two data sets (station data and bike data [18]).

A. Data Collection

This paper uses two data sets, one is historical bike usage data set, the other is station data set. These data sets record data from April 1 to September 30, 2014. Through the statistical collation of the data set, the station information and the historical bike records are unified into bicycle data set, a total of 473,620 records were recorded. The detailed description below can be obtained in Table II.

TABLE II. DETAILS OF THE NEW YORK DATA COLLECTION IN 2014

Bike Data	#Stations	344
	#Bikes	6800
	#Records	5,359,995

B. Baseline & Metric

The method in our work to divide traffic dispatching areas is denoted as Partition Method (PM) by combining K-medoids clustering, association rules and total demand. In order to confirm the effectiveness of our algorithm, we carried out experiments to compare our method with the following baselines:

Bipartite Station Clustering(BSC)[19]. This method grouped individual stations into clusters according to their geographical locations and transition patterns. Finally, the whole traffic is divided into 23 groups.

Adaptive Capacity Constrained K-centers Clustering (CCKC)[20]. This method considers the distance between stations and the location of outliers, grouping outliers with other outliers, and setting up delivery personnel in outliers.

Metric. The metric we adopt to measure results are Sum Of The Squared Errors (SSE).

$$SSE = \sum_{i=1}^k \sum_{p \in C_i} |p - m_i|^2 \quad (5)$$

Where p is the sample point in C_i and m_i is the center of C_i .

C. Experimental result

Station-level partition method. Intuitively, the larger the number of clusters, the lower the prediction accuracy. When there is only one cluster, its usage demand is the whole traffic flow, which can be predicted accurately; when there are clusters, it means that each station forms a cluster, and the outflow/ inflow of the cluster fluctuates greatly, even if it can be predicted, but it is difficult to predict accurately. However, on the other hand, the number of clusters should not be too small, because if the cluster is too large, such as a cluster containing all stations, redistributing bicycles to the cluster cannot provide convenience for users. Therefore, we take the number of outliers as the baseline though many experiments, and finally the number of clusters is determined to be 23.

Similarly, we use another method to evaluate the effectiveness of our method. That's the elbow method SSE we talked about above. When the number of clusters K is less than the number of real clusters, the aggregation degree of each cluster will be greatly increased with the increase of K . When K reaches the real cluster number, the aggregation degree returns will be rapidly reduced with the increase of k , so the decrease of SSE will decrease sharply, and then become flat with the increase of K value. That is to say, the graph of the relationship between SSE and K is as follows: The shape of an elbow, and the K value corresponding to this elbow is the real clustering number of data. Obviously, As can be seen from Figure 5, when the SSE value is the smallest, the number of clusters K is still 23. Therefore, the effectiveness of our method is verified.

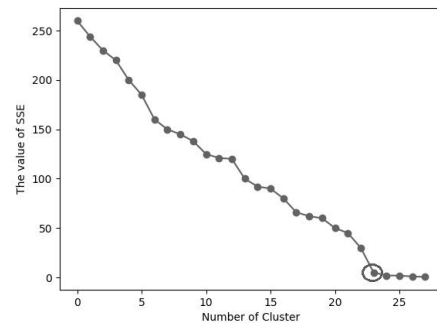


Figure 5. Evaluation of our method

Determination of Minimum Threshold. In addition, when we use Aprior algorithm to further filter the set of strong

association rules, the minimum support threshold will be involved. In the process of borrowing and returning, because some records of data set are abnormal transaction records generated by the manual bicycle dispatching operation of the station, the coefficient of normal bicycle transaction records is taken as $\rho=0.98$; when selecting data samples, there are about 6800 bicycles in the bike sharing system, but a considerable number of bicycles are in the Off-Shelf state during the peak period, that is to say, they do not participate. With the flow of shared bicycles, the coefficient $\omega=0.9$ of mobile bicycles on the rack is taken. when shared bicycles are moving at the station, according to the results of data analysis, bicycles leased from one s may be returned to other 3-5 stations besides their own, so the distribution coefficient of bicycle flow is taken as $\lambda=0.25$, according to formula (2), the minimum support threshold $Sup_{min}=0.1$ is obtained. The final result is obtained through the restraint adjustment of the total regional demand. (as shown in Figure 6.)

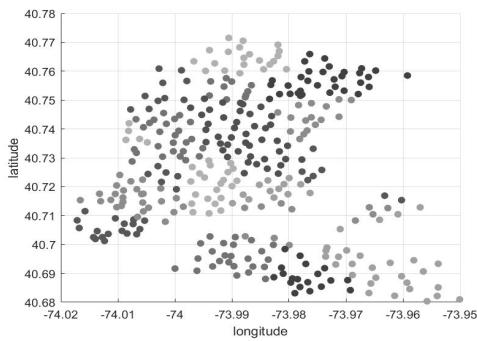


Figure 6. Clustering results

Performance comparison. In order to confirm the effectiveness of our model, we carried out experiments and compared our method with two baselines. CCKC (adaptive constrained central point clustering) and BSC (clustering based on K-means method according to the transformation relationship and geographical location between stations). The effectiveness and efficiency of the proposed PM are shown in Figure 7. It can be seen that for a given number of vehicles, we can concentrate on optimizing the stations and effectively find abnormal stations. With the increase of the number of vehicles scheduled, the number of outlier stations decreases rapidly. PM algorithm can help determine the minimum number of vehicles covering all target stations, or balance operation costs and the number of outlying stations.

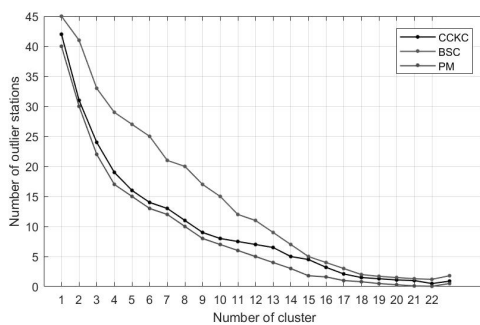


Figure 7. Comparison of clustering efficiency

Clustering Analysis of Shared Bicycle Usage Distribution. Figure 8 shows the demand of shared bicycle in different time at station level and class level. It can be concluded that the trend of shared bicycle demand is more stable after using the zoning algorithm. This demonstrates the effectiveness of our proposed partition method, and also provides the possibility for improving the prediction accuracy of shared bicycle demand in the traffic field.

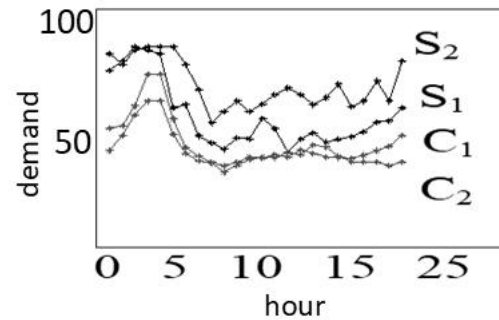


Figure 8. Station and cluster distribution

V. CONCLUSIONS

Based on the characteristics of complex relationship, large scale and large randomness among stations of bike sharing system, we consider the self-mobility of bikes among stations. We study urban bike sharing system with three algorithms: Association rules, K-medoids clustering and total demand restraint adjustment, and take New York City bike sharing system as an example to simulate and partition. This method takes into account the relationship between stations, the attributes of geographical location and the total demand for bicycles. Compared with CCKC and BSC methods, it is concluded that the number of outliers in this method is more stable and the value of SSE is the smallest. Furthermore, The area obtained by our algorithm is more stable than that of single station, which also shows that our method provides a theoretical basis for improving the accuracy of traffic flow prediction.

ACKNOWLEDGMENT

This work is supported by Programs of National Natural Science Foundation of China (No:71461023) .

REFERENCES

- [1] Saberi M , Ghamami M , Gu Y , et al. Understanding the impacts of a public transit disruption on bicycle sharing mobility patterns: A case of Tube strike in London[J]. Journal of Transport Geography, 2018, 66:154-166.
- [2] Li Y , Zheng Y , Zhang H , et al. Traffic prediction in a bike-sharing system[C]// the 23rd SIGSPATIAL International Conference. ACM, 2015.
- [3] Liu J , Sun L , Chen W , et al. Rebalancing Bike Sharing Systems: A Multi-source Data Smart Optimization[C]// Acm Sigkdd International Conference on Knowledge Discovery & Data Mining. ACM, 2016.
- [4] Yang Z , Hu J , Shu Y , et al. Mobility Modeling and Prediction in Bike-Sharing Systems[J]. 2016.
- [5] Li H , Hong L Y , Mo Y C , et al. Restructuring performance prediction with a rebalanced and clustered support vector machine[J]. Journal of Forecasting, 2018(4).

- [6] J. Liu, Q. Li, M. Qu, W. Chen, J. Yang, X. Hui, H. Zhong, and Y. Fu. Station site optimization in bike sharing systems. In ICDM 2015. IEEE, 2015.
- [7] Holmgren J , Moltubakk G , Jody O'Neill. Regression-based evaluation of bicycle flow trend estimates[J]. Procedia Computer Science, 2018, 130:518-525.
- [8] Dell'Amico, Mauro, Iori M , Novellani S , et al. A destroy and repair algorithm for the Bike sharing Rebalancing Problem.[J]. Computers & Operations Research, 2016, 71(C):149-162.
- [9] Zhang J , Pan X , Li M , et al. Bicycle-Sharing System Analysis and Trip Prediction[J]. 2016.
- [10] Longbiao Chen, Daqing Zhang, Leye Wang, Dingqi Yang, Xiaojuan Ma, Shijian Li, Zhaohui Wu, Gang Pan, Thi-Mai-Trang Nguyen, and Jérémie Jakubowicz. 2016. Dynamic cluster-based over-demand prediction in bike sharing systems. In Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing. ACM, 841–852.
- [11] Mike Benchimol, Pascal Benchimol, Benoît Chappert, Arnaud De La Taille, Fabien Laroche, Frédéric Meunier, and Ludovic Robinet. 2011. Balancing the stations of a self service “bike hire” system. *RAIRO-Operations Research* 45, 1 (2011), 37–61.
- [12] Gerardo Berbeglia, Jean-François Cordeau, Irina Gribkovskaia, and Gilbert Laporte. 2007. Static pickup and delivery problems: A classification scheme and survey. *TOP* 15, 1 (2007), 1–31.
- [13] Lin L , He Z , Peeta S . Predicting Station-level Hourly Demands in a Large-scale Bike-sharing Network: A Graph Convolutional Neural Network Approach[J]. 2017.
- [14] Faghih-Imani A , Hampshire R , Marla L , et al. An empirical analysis of bike sharing usage and rebalancing: Evidence from Barcelona and Seville[J]. *Transportation Research Part A: Policy and Practice*, 2017, 97:177-191.
- [15] Zhang L , Tang S , Yang Z , et al. Demo: Data Analysis and Visualization in Bike-Sharing Systems[C]// International Conference on Mobile Systems. ACM, 2016.
- [16] Oliveira G N , Sotomayor J L , Torchelsen R P , et al. Visual Analysis of Bike-Sharing Systems[J]. *Computers & Graphics*, 2016, 60:119-129.
- [17] Bouveyron C , Côme, Etienne, Jacques J . The discriminative functional mixture model for a comparative analysis of bike sharing systems[J]. *Dissertations & Theses - Gradworks*, 2016, 9(4).
- [18] <http://www.citibikenyc.com/system-data>.
- [19] Chen T , Lu S . Robust Vehicle Detection and Viewpoint Estimation with Soft Discriminative Mixture Model[J]. *IEEE Transactions on Circuits and Systems for Video Technology*, 2016:1-1.
- [20] Li Q L , Chen C , Fan R N , et al. Queueing Analysis of a Large-Scale Bike Sharing System through Mean-Field Theory[J]. 2016.
- [21] Tomaras D , Boutsis I , Kalogeraki V . Lessons Learnt from the analysis of a bike sharing system[C]// International Conference. 2017.
- [22] O'Mahony E , Shmoys D B . Data Analysis and Optimization for (Citi)Bike Sharing[C]// Twenty-ninth Aaai Conference on Artificial Intelligence. AAAI Press, 2015.
- [23] Liu J , Li Q , Qu M , et al. Station Site Optimization in Bike Sharing Systems[C]// 2015 IEEE International Conference on Data Mining (ICDM). IEEE Computer Society, 2015.
- [24] Basch C H , Ethan D , Zybert P , et al. Public Bike Sharing in New York City: Helmet Use Behavior Patterns at 25 Citi Bike™ Stations[J]. *Journal of Community Health*, 2014, 40(3):530-3.
- [25] Chen Q , Sun T . A model for the layout of bike stations in public bike-sharing systems[J]. *Journal of Advanced Transportation*, 2015, 49(8):884-900.
- [26] Vogel P , Saavedra B A N , Mattfeld D C . A Hybrid Metaheuristic to Solve the Resource Allocation Problem in Bike Sharing Systems[J]. 2014.
- [27] Li Z , Zhang J , Gan J , et al. Large-Scale Trip Planning for Bike-Sharing Systems[C]// 2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS). IEEE Computer Society, 2017.
- [28] Bao J , Xu C , Liu P , et al. Exploring Bikesharing Travel Patterns and Trip Purposes Using Smart Card Data and Online Point of Interests[J]. *Networks and Spatial Economics*, 2017, 17(4):1231-1253.
- [29] Bordagaray M , dell'Olio, Luigi, Fonzone A , et al. Capturing the conditions that introduce systematic variation in bike-sharing travel behavior using data mining techniques[J]. *Transportation Research Part C: Emerging Technologies*, 2016, 71:231-248.
- [30] Chen L , Xiaojuan M A , Nguyen T M , et al. Understanding bike trip patterns leveraging bike sharing system open data[J]. *Frontiers of Computer Science*, 2017, 11(1).
- [31] Cagliero L , Cerquitelli T , Chiusano S , et al. Predicting critical conditions in bicycle sharing systems[J]. *Computing*, 2017, 99(1):39-57.
- [32] Zhang J , Yu P S . Trip Route Planning for Bicycle-Sharing Systems[C]// 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC). IEEE, 2016.
- [33] Faghih-Imani A , Anowar S , Miller E J , et al. Hail a cab or ride a bike? A travel time comparison of taxi and bicycle-sharing systems in New York City[J]. *Transportation Research Part A Policy and Practice*, 2017, 101:11-21.
- [34] Zhang J , Pan X , Li M , et al. Bicycle-Sharing System Analysis and Trip Prediction[J]. 2016.
- [35] Benarbia T , Labadi K , Darcherif A M , et al. Real-time inventory control and rebalancing in bike-sharing systems by using a stochastic Petri net model[C]// International Conference on Systems & Control. 2018.

Anomaly Detection in the Registry of the Secondary Energy Distribution Network

1st Carlos Fonsêca

Post-Graduation Program in Computer Engineering
University of Pernambuco
Recife, Brazil
ccbf@ecomp.poli.br

2nd Alexandre Maciel

Post-Graduation Program in Computer Engineering
University of Pernambuco
Recife, Brazil
amam@ecomp.poli.br

Abstract—The paper aims to create an intelligent model of data analysis in the registry of the secondary energy distribution network. With emphasis on the search for possible inconsistencies that can be only cadastral or really physical. For this, it uses some techniques of data mining giving focus for the detection of anomalies. This study used a private database containing information about the assets that make up the secondary energy distribution network, such as: poles, transformers, disconnectors, among others. The research was developed following all steps presented in the CRISP-DM methodology. To detect the anomalies, it was used algorithms Isolation Forest, DBSCAN and BIRCH. As a result, the three algorithms pointed to a set of specialty anomalies, validated by a specialist, however, Isolation forest was more accurate in the inference of the anomalies. From this study, distribution companies will be able to identify risky or financially problematic situations in advance.

I. INTRODUCTION

With the great volume of data coming from diverse sources and produced by different entities (citizens, applications, public institutions, among others), an urgent need was generated to treat this data in an automated way. Through the extraction of information and production of knowledge, applications can be created that directly reflect the improvement of services available to all citizens. In response to these needs, emerged Data Mining (DM) [1].

Data mining is a technology that has emerged from the intersection of three areas: statistics, artificial intelligence, and machine learning. It aims to extract useful knowledge (eg patterns) of complex and vast data. The techniques used for data mining are from different approaches and their applications depend on the nature of the data and the scenario of the problem [2].

There are six categories of widely accepted and implemented data mining techniques: classification, grouping, association, regression, summarization, and anomaly detection [3].

Anomaly detection techniques have been applied successfully in critical systems and result in better damage control and component failure prediction [4].

Therefore, the accomplishment of this work becomes significant as it seeks to contribute with studies for the detection

of anomalies, in addition to studies in the area of data mining in the energy sector.

II. APPROACHES BASED ON ANOMALIES DETECTION

A. Detection of Anomalies

With the increasing use of advanced database technologies developed over the last few decades, it is not difficult to efficiently store huge volumes of data and be able to retrieve them whenever necessary. While data storage is very valuable to an organization, most organizations are unable to extract relevant information in a timely manner for decision making. This situation has aroused recent interest in research in the area of data mining [7].

Detecting behavioral deviations in data can solve problems of identifying frauds, disturbances and irregularities in general. This works based on the identification of points placed outside reasonable limits, called anomalies [8].

An anomaly is an observation that presents a great distance from the others of the series sampled or that is inconsistent. The existence of anomalies typically implies the interpretation of the results of the statistical tests applied to the samples [9].

Several applications today require in-depth data analysis to filter sporadic values and ensure system reliability. Such techniques are especially useful for fraud detection, where malicious attempts often differ from most nominal cases and can therefore be prevented by identifying external data. These anomalies can be defined as observations that deviate sufficiently from most observations to consider that they were generated by a different process [10].

These observations are called anomalies when their number is significantly smaller than the proportion of nominal cases, usually less than 5% [10].

Various methods of learning machines are suitable for anomaly detection. However, supervised algorithms are more restricted than unsupervised methods because a set of labeled data needs to be provided. This requirement is particularly expensive when labeling must be performed by humans. Dealing with a fairly unbalanced class distribution, which is inherent in anomaly detection, can also affect the efficiency of supervised algorithms [11].

III. METHODOLOGY

One of the most popular methodologies to increase the success of data mining processes is the CRISP-DM (Cross-Industry Standard Process for Data Mining) [12]. The methodology defines a non-rigid six-phase sequence that allows the construction and implementation of a mining model to be used in a real environment, helping business decisions [13]. Therefore, the development of this work will follow the six phases of CRISP-DM.

A. Business Understanding

Electric power distribution networks are composed of high, medium and low voltage lines. These are, respectively, the sub-transmission networks, primary and secondary energy, where they concentrate different purposes. The secondary network of distribution, focus of analysis in this work, is that which supplies the common users and small building facilities, that is, the great mass of users of the distribution companies [5].

Every year, hundreds of people are hit by direct or indirect damages from clandestine electricity connections throughout Brazil. There are records of transformer overload that causes harm to the community and fires that are traced back to this type of crime [6].

B. Data Understanding

To obtain the partial results, it was necessary to integrate the bases provided by the company into a single table, containing all the information passed for analysis. After the integration, some necessary techniques were applied for the selection of the data that will compose the model.

The databases were integrated using one table as reference for joining the others. The table used as reference for the merge was the one referring to the poles that make up the secondary distribution network. At the end of this initial integration a table with 165 attributes and 2,249,159 records was obtained. It can be noted that this is a table with large horizontal dimension and possibly reducible. Some analyzes were carried out in order to obtain a possible reduction of dimensionality.

C. Data Preparation

Initially, a statistical analysis of the attributes was performed, which served to deepen the data. It was possible to verify the quantity, mean, standard deviation, minimum and maximum of each quantitative attribute.

An analysis was performed regarding the variance of the attributes, since attributes with low variance do not influence significantly in the models of machine learning. Soon after the first adopted strategy a reduction was achieved for 112 attributes.

Next, an analysis was performed on the correlation of the variables, where it was possible to observe that only 47 variables had some degree of correlation with some other one in question. Among the variables with some degree of correlation, few had a correlation with greater significance.

After the previous step was created a data dictionary that helped a little more in the understanding of the data. With

the data dictionary, it was made explicit that most attributes are categorical and discrete quantitative. With a deeper understanding of the data it was possible to perform a new dimensionality reduction for 96 attributes.

After consulting with a specialist in the area of electricity, some more attributes were removed, as it was clear to the expert that they would not add value to the model. At the end of this attribute selection step, the database was reduced to 52 attributes.

1) *Sub-databases*: It was necessary to create sub-databases, in which they were extracted from data from the main database, which contained 2,240,756 records and 52 attributes. The extraction of sub-bases was necessary because some of the algorithms used were not having good results with the complete base and also to obtain a better understanding about the influence of each equipment.

The sub-databases were extracted as follows: base only with the attributes of the poles, base only with poles that had some equipment connected to it, base with poles only with capacitor banks, base with poles only with fuses, base with the poles with only lightning rods, base with the poles only with disconnectors, base with the poles only with sectionalizers and a base with the poles only with transformers.

These sub-databases were important for a better understanding of the influence of each equipment in the general context of the problem.

D. Modeling

The modeling implementation will be performed using the Python platform, as it provides a variety of routines for statistical calculations (linear and nonlinear methods, statistical tests, grouping, simulations, anomaly detection, etc.) and is strongly recommended for modeling and analysis statistic.

E. Evaluation

The evaluation was carried out mainly by the specialist engineer of the electric sector. The three algorithms were confronted in their results, with the focus of a more robust validation regarding the performance of each of them.

F. Deployment

An application, for use by electrical industry experts, is still being developed. Upon completion of the development, the tool will be thoroughly tested in real production scenarios.

IV. RESULTS AND DISCUSSIONS

A. Isolation Forest

The implementation of Isolation Forest [14] that was used to obtain the results of this research came from the scikit-learn library [15]. The library provides the results on a scale between -1 and 1, with the closest results of -1 being characterized as possible anomalies. The values of the parameters that were configured in the execution of the algorithm are: max_sample = auto and contamination = 0,01.

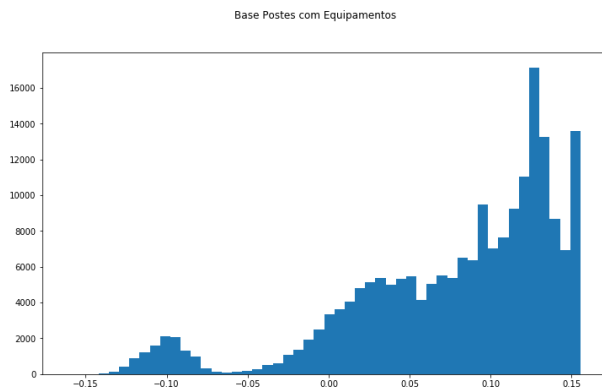


Fig. 1. Histogram showing the result of the execution of the Isolation Forest to the database of poles that have some equipment.

In the graph of Figure 1, the possible anomalous points in the region between -0.15 and -0.10 can be found. The other points of the base were condensed to the right of the graph, indicating that they were points more difficult to isolate and therefore have a high probability of not being anomalies.

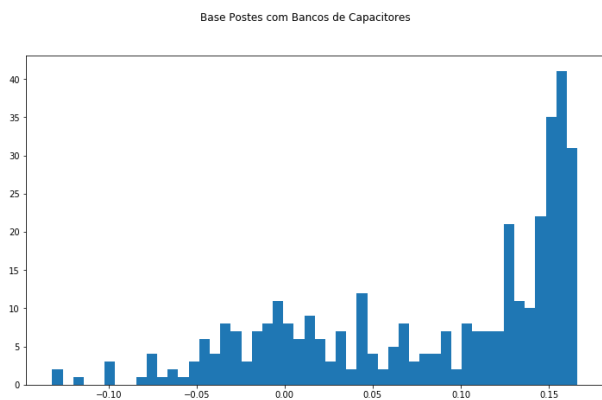


Fig. 2. Histogram showing the result of the execution of the Isolation Forest to the database of the poles that have capacitor banks.

In Figure 2, it is noted that the points received normality notes in distinct groups and not in a continuous block. The points with the highest anomalous characteristics were well isolated and closer to -0.10 or less.

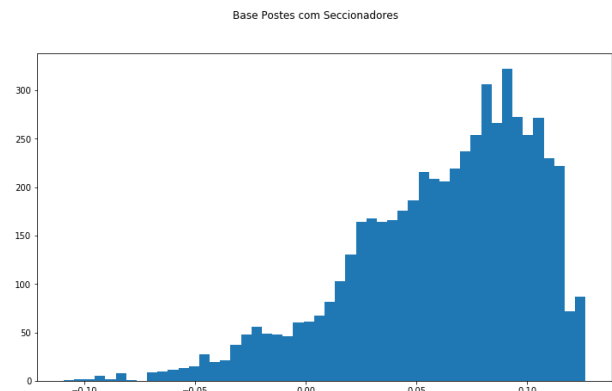


Fig. 3. Histogram showing the result of the execution of the Isolation Forest to the database of the poles that have disconnectors.

In Figure 3, two isolated groups located to the left of the graph are evident. One of the groups of points is a little closer to the normal characteristics to the right of -0.10. The most extreme group and also with fewer points is that it appears to have more anomalous characteristics.

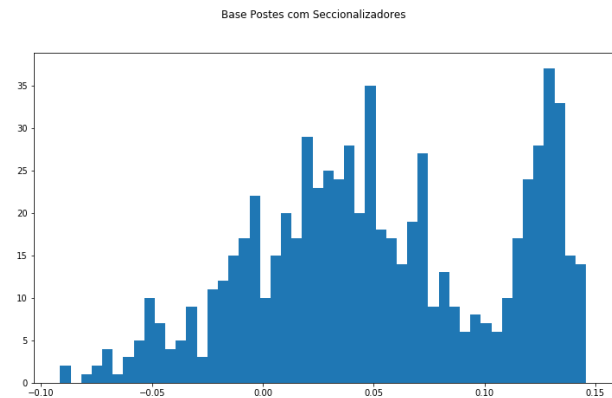


Fig. 4. Histogram showing the result of the execution of the Isolation Forest to the database of the poles that have sectionalizers.

In Figure 4, you may see a small group of possible anomalous points located to the left of the graph. The rest of the data were grouped in points closer to normal.

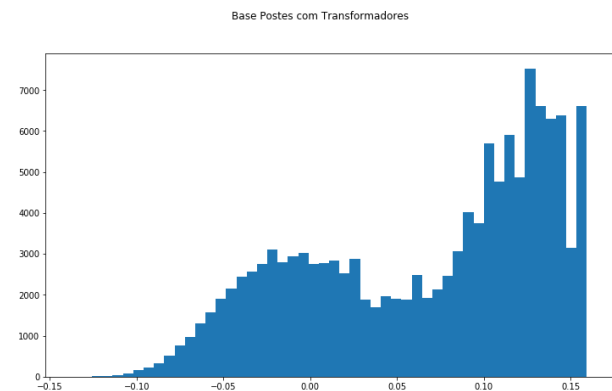


Fig. 5. Histogram showing the result of the execution of the Isolation Forest to the database of poles that have transformers.

In Figure 5, can be analyzed a very small group to the left, far from the rest of the points. This group has a great possibility of being anomalous points.

With emphasis on the enhancement of Isolation Forest [14] results, two other algorithms, DBSCAN [16] and BIRCH [17], were used. The three algorithms used are part of the Scikit-Learn [15] library. The algorithms DBSCAN [16] and BIRCH [17], had their results compared to the results of Isolation Forest [14], in order to provide greater certainty about the anomalies found.

B. Analysis of Anomalies

With the result of the execution of the Isolation Forest [14] algorithm, in the sub-databases, being supported by the specialist engineer and the algorithms DBSCAN [16] and BIRCH [17], several anomalies.

1) *Poles with Capacitor Banks*: In this sub-base anomalies were found by the algorithms and a specific one was validated by the specialist engineer. The capacitor bank, which the specialist validated, was the only one in the whole sample that had a different cell power than the others.

2) *Poles with Fuse Keys*: In the sub-base in question, an anomaly was found and validated by the specialist. Anomaly analysis demonstrated that the pole on which the fuse switch was located demonstrated a single supported stress within all samples.

3) *Poles with lightning rods*: The analysis of the sub-base of the poles with lightning, presented a single para-ray that was with the primary tension different from the others.

4) *Poles with Disconnectors*: Among the poles with disconnectors that presented anomalies, the one that stood out showed the highest rated current among all.

5) *Poles with Sectionalisers*: All the anomalous poles that contained sectionalizers presented the same nominal voltage, being the only one different in this aspect which demonstrated greater anomalous characteristics.

6) *Poles with Transformers*: Among the poles of this sub-base, two main anomalies were found, later validated. The first is that there are two poles with heights and the same construction materials, but with different efforts. And the next anomaly, comes from a single pole with a state different from the others.

V. CONCLUSION

This article focuses on a major challenge: the detection of anomalies in the registration of the secondary energy distribution network. It was concluded that this work was able to meet the challenge, after the identification of the anomalies by the algorithms and subsequent results of the dosages by a specialist engineer in the electrical sector.

The identified anomalies can be complex and difficult to understand we argue that anomaly detection approaches should support experts when analyzing anomalies along with the related alarms. It is shown how much information can be provided by the proposed rule based anomaly detection approach.

We assume that it is necessary to identify but also to understand anomalies to choose appropriate anomaly countermeasures. Future work will concentrate on expanding and evaluating the proposed approaches root cause analysis capabilities. For this, visualization and management tools will be created that enable to handle the provided information in an interactive manner.

VI. ACKNOWLEDGEMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior - Brasil (CAPES) - Finance Code 001.

We would also like to thank the financial and technological support of In Forma Software company.

REFERENCES

- [1] Goldschmidt, R., Bezerra, E., Passos, E. Data mining: Conceitos, técnicas, algoritmos, orientações e aplicações. Rio de Janeiro-RJ: Elsevier, 56-60. (2015)
- [2] Kampff, Adriana Justin Cerveira. Mineração de dados educacionais para geração de alertas em ambientes virtuais de aprendizagem como apoio à prática docente. (2009).
- [3] Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine* 17.3 (1996): 37.
- [4] Worden, Keith, Graeme Manson, and Nick RJ Fieller. "Damage detection using outlier analysis." *Journal of Sound and Vibration* 229.3 (2000): 647-667.
- [5] ABRADÉE. A Distribuição de Energia. <http://www.abradee.com.br/setor-de-distribuicao/a-distribuicao-de-energia>. Last accessed 16 Nov 2018
- [6] G1. Mais um transformador pega fogo e prejudica lojistas em Itapetininga. <http://g1.globo.com/sao-paulo/itapetininga-regiao/noticia/2014/02/mais-um-transformador-pegafogo-e-prejudica-lojistas-em-itapetininga.html>. Last accessed 16 Nov 2018
- [7] Lu, Hongjun, Rudy Setiono, and Huan Liu. Neurorule: A connectionist approach to data mining. *arXiv preprint arXiv:1701.01358* (2017).
- [8] Oliveira, Cledson D., et al. Detecção de Fraudes, Anomalias e Erros em Análise de Dados Contábeis: Um Estudo com Base em Outliers. *Revista Eletrônica do Departamento de Ciências Contábeis & Departamento de Atuária e Métodos Quantitativos (REDECA)* 1.1: 102-127.
- [9] Barnett, Vic, and Toby Lewis. *Outliers in statistical data*. Wiley, 1974.
- [10] Domingues, Rmi, et al. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition* 74 (2018): 406-421.
- [11] Japkowicz, Nathalie, and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis* 6.5 (2002): 429-449.
- [12] Chapman, Pete, et al. *CRISP-DM 1.0 Step-by-step data mining guide*. (2000).
- [13] Moro, Sergio, Raul Laureano, and Paulo Cortez. Using data mining for bank direct marketing: An application of the crisp-dm methodology. *Proceedings of European Simulation and Modelling Conference-ESM'2011. EUROSIS-ETI*, 2011.
- [14] Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. 2008 Eighth IEEE International Conference on Data Mining. IEEE, 2008.
- [15] Pedregosa, Fabian, et al. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12.Oct (2011): 2825-2830.
- [16] Ester, Martin, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. *Kdd*. Vol. 96. No. 34. 1996.
- [17] Zhang, Tian, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. *ACM Sigmod Record*. Vol. 25. No. 2. ACM, 1996.

Analyzing the impact of Technological KM and Participatory KM in FTA

Diego Cardoso Borda Castro¹, Carlos Eduardo Barbosa^{1,2}, Luis Felipe Coimbra Costa¹, Jano Moreira de Souza¹

¹COPPE – Graduate School and Research in Engineering
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
{diegocbcastro, eduardo, luisfcosta, jano}@cos.ufrj.br

²CASNAV – Center for Naval Systems Analyses
Brazilian Navy
Rio de Janeiro, Brazil

Abstract — The labor market is becoming increasingly competitive, with new technologies and products being launched at all times. Companies that have the privilege of creating the products at the beginning of innovation can serve more customers and thus generate more profits. In this search for the market, companies are increasingly willing to please their customers, trying to understand what they seek. There are several concepts, ways, strategies, and technologies that help companies better understand what people are searching for, interfering with this market. These strategies include Future-oriented Technology Analysis (FTA) and others, such as Knowledge Management (KM), from which we focus on Technological and Participatory practices. These strategies are directly related bringing a fundamental advantage to the company that knows how to use these concepts effectively.

Keywords — Knowledge Management, Participatory Knowledge Management, Technological Knowledge Management, Future-oriented Technology Analysis.

I. INTRODUCTION

The last years have shown a growth in new ways of thinking through information technologies. With this growth, also increased the competition for the manufacture of products where companies are seeking to meet more and more the needs and expectations of their customers, leading to a race for innovation. All these transformations make companies do not know where, when and how to innovate to become more competitive ahead of their competitors.

A company needs to have a competitive intelligence to become more attractive to its consumers, knowing when to make deliver the right product at the right time. Therefore, innovating in management methods and processes is one of the main challenges of a market with fierce competition, a high degree of uncertainty and a great deal of information available.

As a possible solution to this problem, the prospection is introduced to increase the company's competitiveness, seeking to find new trends so that companies can know where they should invest. Future studies directly support decision-making at various levels in the society. The purpose is not to predict the future, but find desirable futures and ways to achieve them.

Future-oriented Technology Analysis (FTA) is an umbrella term for a wide range of activities that facilitate decision-making and coordinated action, especially in the formulation of science, technology and innovation policies. Understanding the changes in technologies is only a part of the understanding of the market; some other aspects can help the companies to gain their competitiveness. Among them, there are the

Technological Knowledge Management (TKM) that consists of numerous practices and use of new tools and the Participatory Knowledge Management (PKM), which involves the sharing of information through groups of people, social networks, and institutions. These strategies may be directly related, bringing advantages to the company that uses them effectively.

II. THEORETICAL FOUNDATION

A. Future-oriented technology analysis (FTA)

Drucker [1] states that it is not important to predict the future because it considers that it is uncertain and it is not possible to be sure of what is to happen. In contrast, Schwarz [2] argues that several managers can already understand the need to study the future for understanding what is needed.

A company must always be ahead of its competitors in technological innovations, being aware of the market's directions and prepared to face or take advantage of new technologies [3]. To make it happen, the FTA is introduced, and its importance is undeniable. However, FTA has many different concepts which are still being studied and deepened.

FTA can be defined as a set of methodologies to support decisions about emerging technologies, including their development and future impacts [4]. Therefore, several groups with different goals use a set of several approaches for the future that share some assumptions and differ in others. Their objective is to support decision-makers with analyses and new ideas to be prepared for the future [5].

B. Technological Knowledge Management (TKM)

Knowledge Management requires technologies to be done efficiently: they support strategies, processes, and methods that help to better disseminate and apply information within the enterprise. Several technologies that can help KM such as the implementation of intranets, data warehouses, data mining, decision support tools, video conferencing, groupware, electronic panels, online databases, expert systems, intelligent search agents and management of electronic documents.

TKM seeks to understand technological progress and its impacts, to enable institutions to deal with change and, above all, to integrate innovation into organizational strategy [6]. Currently, it is considered to have the most significant impact on the changes that are taking place within companies; however, although technology is widely recognized as being essential for competitiveness, technology management has been one of the most challenging activities among the attributions to managers [7]. Despite the advantages offered by

technologies, some challenges are faced by companies when trying to implement them, Betz [7] cites some such as: "the not invented here syndrome", the physical separation of the research laboratories of the responsible sector, the use of inadequate techniques of planning and technological monitoring, being able to result in a distorted vision about the future and the late use of new technologies, resulting in the loss of the market for competitors [7].

C. Participatory Knowledge Management (PKM)

PKM is the KM derived to participatory management, which is the set of organizational conditions and managerial behaviors that encourage the participation of all in the management process. Participatory management empowers the group members to make organizational decisions [8]. Involving of everyone means that, in the beginning, no person, at any hierarchical level, should be excluded from the participatory process, everyone is aware of the methods and approaches that are being implemented within the company.

A more knowledge-oriented view can be given by Valtolina et al. [9], where they define PKM as the use of methods, tools, and guidelines to mediate cooperation between user groups and IT professionals. PKM facilitate collaboration between users who have different types of knowledge, and each user is responsible for disseminating the information. In general, PKM can be summarized as the use of a group of people to generate and use knowledge on different topics.

III. USING TKM, PKM AND FTA IN DECISION-MAKING

A. FTA relation to PKM and TKM

Godet [10] argues that all those who intend to foresee the future (prophets, oracles, seers, sensitives, clairvoyants, and others) are impostors because the future does not exist, it is yet to be created and therefore it is not written anywhere. In fact, talking about technological prospecting regarding knowledge can be a bit tricky as the future is uncertain and can change at any time. So how can there be statements about such? FTA seeks to assist in decision-making by looking at a history of events that may have implications for future circumstances. Logically it is not yet possible to predict the future and to know for sure what will happen, but it is possible to look at the past and the present to raise hypotheses and assumptions about some events of the future.

The junction of the FTA and KM can be seen as information-based activities, to enable an organization to anticipate its competitors and anticipate the changes that may occur, and as a consequence take advantage of that condition to increase its competitive intelligence [11]. FTA has a long-term vision and a greater focus on technology, while competitive intelligence has a short-term view with a broader discussion.

Bell and Olick [12] reformulate the idea of knowledge of the future by arguing that it is possible to postulate and think to predict some possibilities. These postulates can be seen as sets of knowledge that have to do with the consequences of situations that can evolve into a possible future.

The FTA-KM relationship is relevant since the former tries to predict future events while the latter focuses on knowledge management within the company. If the next trends are

incorporated in the KM before they happen, they can be introduced as soon as possible in the company, bringing a competitive intelligence that aims at finding the risks and opportunities in the market early, to generate a competitive advantage. The combination of these two ideas can be seen in a very simplified way, as if FTA were to seek new knowledge and trends that can affect the market, and then enters the part of the KM that will analyze this information to manage new expertise within the company, trying to understand, store and use in the best possible way what was introduced.

As already explained, KM has some ramifications, such as TKM and PKM, and each of them relates to FTA differently, each with its particularities. The basis for the intermediation between the TKM and FTA starts at the information collection stage, which is then addressed to the treatment of them, and, finally, the systematization of the information. In other words, it is the analysis, interpretation, and production of knowledge so that, in the end, the dissemination of results is done. The TKM and PKM are interconnected and are two poles of different concepts of KM, the former more geared towards the use of technologies and the latter for the active participation of all in an interactive way.

One of the areas of FTA that can integrate more with TKM is the technical evaluation, which seeks to predict the consequences of the introduction of a specific technology in all the spheres with which it interacts. Looking at technology management, it aims at understanding the use of tools and their impacts to enable institutions to cope with change. By reading the two concepts, it is easy to see that there is a relationship between the two, where they can help each other. With these ideas in mind, prospecting can seek new technologies that can be supported by TKM, where the former finds new concepts that can be integrated into the organization while the latter tries to manage these concepts better, trying to find better ways to apply each one of the ideas more effectively. Overall, the first aid provided by TKM for knowledge management is the use of new technologies to facilitate the exchange of information between users, increasing data speeds, assisting in connecting people of different groups.

Jaspers et al. [13] state that in the TKM, the stakes are the ways of dealing with concrete facts, for example, sites, databases, publications, dissemination procedures, among others. In this concept, forecasting processes deal with a considerable mass of information and rely primarily on systems to obtain more real and comprehensive knowledge. In this part, new technologies can help in the efficient and faster dissemination of information that has been constructed through FTA methods. On the other hand, FTA may be able to find some relevant information about the use of possible technologies, which now need to be learned and implemented within the company.

When speaking in a group of people, the first thing that comes to mind is the PKM, which has more emphasis on social practices and interactions between individuals and groups that create, develop and use knowledge, approaching the idea of the social learning cycle, taking into account cultural factors, values, and opinions. PKM includes various means of exchange and communication, such as a face-to-face meeting.

Surveys are increasingly using the dynamics, effects of simultaneous effort/cooperation of groups and processes of dialogue to achieve new knowledge [13]. This is one of FTA main points of connection with PKM, the use, and application of information that is provided by groups of people to create new concepts within companies, linking people who are positioned in different groups.

As is the case in many other KM fields, future-oriented TKM concepts run the risk of being considered sufficient, neglecting the social side of PKM. However, a forecast is not only a process for collecting, analyzing and filtering data, and it also has a social dimension that is also very important: society needs to exchange ideas and develop visions for a collective future [13]. One of the challenges of PKM is how to efficiently use participants' knowledge and imagination to search for hypotheses for the future, by continually initiating and optimizing the interaction of information and people [13].

One of the areas of FTA that best fit with the PKM is the vision that guides the future by consensus, which uses as a base the opinions collected through the cognitive and intuitive process of a group of experts, and the PKM aims at using the mutual knowledge of a group of people. A link between these two concepts is easy to see, and the participation of groups can help create new ideas for the future.

B. Proposed Approach

As it was seen, PKM and TKM are directly related to FTA and bring several benefits to the company that correctly uses these techniques, but it is worth remembering that with these advantages they bring some harm. So, we need to know how to use them correctly. The main problems encountered were: the fear of using new technologies, the reluctance on the part of the professional to share their knowledge and the time of learning, depending on the technology presented. We seek to solve these problems creating a new concept junction model (PKM, TKM, and FTA), improving management agility, decision-making efficiency and the generation of competitive advantage.

The proposed model is presented in Fig. 1. The starting point for this methodology will be to look from the past to the present to develop a future forecast. In this method, two metrics are used: a projection based on existing data (looking at the past) and one based on queries and opinions (having something more focused on people and the present) so that, thus the future can be predicted. These two methods are being used to try to reduce the number of predicted errors and have both advantages and disadvantages. The first is a quantitative method and requires reliable and standardized historical bases, while the second is a qualitative method that presents problems with the limit of knowledge of specialists and interested people. With the two being used collaboratively, it is hoped that the problems cited will be minimized.

After the initial presentation of the methodology making use of FTA, some techniques are necessary for the forecasts to be made in the best way. Some are presented below.

- **Market monitoring:** used to predict possible trends, be it short, medium and long-term;
- **Interviews and brainstorming:** collecting participants' opinions;

- **Cross-impact analysis:** understanding the relationships of factors and trends found;
- **Roadmaps:** analysis of the technologies found;
- **Creation of scenarios:** considerations of plausible futures.

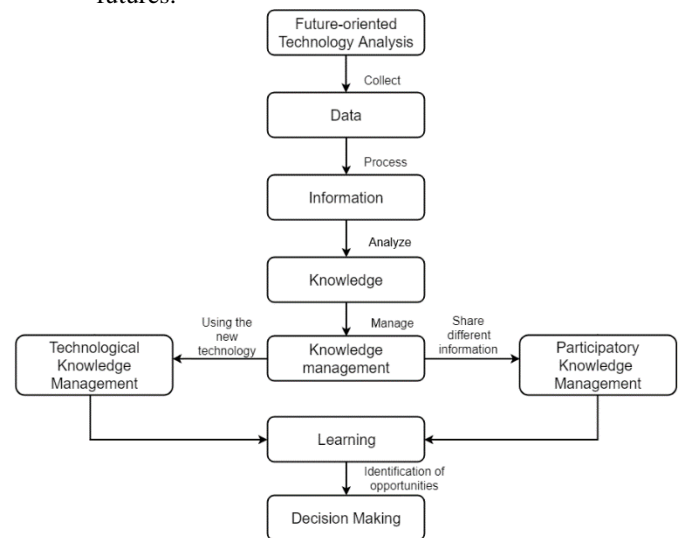


Fig. 1. Junction model with concepts of FTA, PKM, and TKM.

Five prospecting techniques have been presented, each of which has advantages and disadvantages. The use of trends offers predictive substances based on quantifiable parameters in the short term; however, at the same time, it requires historical data and thus vulnerable to change; the use of interviews and brainstorming can bring a large number of information depending on the way it is performed, yet, it is limited to the knowledge of the people who participated in the process; the analysis of cross-impacts is of great importance when a critical scenario is being developed, since it can understand the relationships between the information that has been found, but care must be taken not to obtain a biased and unreal result; the use of roadmaps seeks to implement and execute strategic maps in order to align the company's strategy with its technological capabilities, but care must also be taken in this part to avoid constructing something that cannot be accomplished; Finally, the creation of scenarios presents rich and complex portraits of possible futures, but it may bring something unattainable. Practitioners must observe which methodology best fit their context, and it is worth emphasizing that a combination of techniques is always well-seen, aiming at reducing the number of future errors. In this phase the first stage of the process ends, bringing with it the new technologies that have been recovered through prospecting. With this, many new and old data will be collected (loose data or random data without any analysis). After this collection, the data will be analyzed and thus transformed into information (organized data), and finally, this information will be converted into knowledge entering the context of the company.

With the knowledge in hand, steps in the KM. As said before, it works on the data that the company has, managing knowledge. In this approach, this management will be divided into two parts, the control of technological expertise and the participatory management of experience, each with its function. The former will try to understand the technical

culture that was brought by FTA so that they can be introduced into the company. At this stage, two problems that have already been described are added; the fear of using new technologies and the lengthy learning time depending on the technology presented. In order to solve these two problems, four different approaches will be used: the first step will be to make a conceptualization of the employees, if they seek to understand the benefits of the technologies will be easier to accept them, as a consequence, the person will also be more willing to learn to use it. Patience, therefore, is essential in this first moment, it is necessary to explain calmly and repeat the information several times, if necessary. This first point will help the employee to accept the technology. Then, a series of training is started which can be done on the person's machine, leaving the user with the highest possible tranquility. On the other hand, face-to-face training will also be carried out, where the person can take his most recurring questions to be clarified with specialized professionals in the field. In the third stage, the concept of pair programming will be used with another view, where some jobs with different levels of difficulty will be advised to various people who have done the training, always seeking to leave a more experienced person with a less skilled one. In the fourth and final phase, it will be suggested that people who have never performed a particular task can try, to be able to learn more about the work and, consequently, decrease the dependency of specialists in a specific area within the company. This last proposal would help the employees to create, in turn, a desire to seek new knowledge, because only then would they be able to deliver the task.

Once the TKM issues have been resolved, PKM is introduced so that people from many different knowledgeable areas can share their information in a unified way. The main idea is everyone would participate equally in this step, being able to collaborate with what they judge necessary, completely altering the organizational climate, where all employees would be like "leaders" having a self-management, stimulating, guiding and coordinating changes. This model is very flexible and aims at leveling the group, boosting participation and dismembering the traditional model. For this model to be deployed correctly, two points must be aligned within the company. The first one is the need for communication between the sectors to make information exchange possible. The second is that a friendly working environment is necessary, where everyone can be allowed to speak equally.

In the PKM stage, a new problem is presented, the reluctance on the part of the professional in sharing their knowledge. In the search for the solution of this problem, some ways of exchanging information will be presented. The first is to create an intranet so that employees can register what they find relevant. The second is to create a bank of ideas to encourage employees to adopt a creative attitude. Awards would be given to the ideas that were judged more pertinent, promoting a friendly competition among the participants. The last thought would be the creation of study groups within the company, where groups with specific themes would be created, each week a different group would talk about what they have learned over time, thus spreading the information in another way. The groups would be formed with a fixed time, so that people are always participating in different groups at all times,

sharing what they learned. This completes the stages of the search, analysis, knowledge generation and conceptualization. Learning in this scheme can be seen as an extension of the idea of experience, i.e., people will do with everything they have understood and learned. They know how to use the knowledge in the right way, identifying what is the best decision to make.

IV. FINAL REMARKS

The market is increasingly competitive with new creations. For a company to win its market, it must have an advantage over the others. In this quest for advantages, technological prospecting methods are being used so that it is possible to create visions of the future and thereby to know what are the new preferences and trends of this market. In this search for new technologies, FTA can benefit from several methods to try to predict the future, such as the use of road mapping, expert opinion, trend monitoring, among many others.

Once this new information has been retrieved and stored, it needs to be handled and managed for better use. At that time the part of the KM enters, seeking to manage this data the best way. For this to happen effectively, KM was branched out in two areas to PKM and TKM, each with its primary focus.

In this work, we presented a brief explanation about the aspects of FTA, TKM, and PKM, after the necessary reports have been made, a search on what has already been done to improve this area was carried out, and finally a new approach of utilization of these three concepts. As future work we intended to test it in an organization to confirm its viability.

REFERENCES

- [1] P. F. Drucker, *The Essential Drucker*. HarperCollins, 2001.
- [2] J. Oliver Schwarz, "Assessing the future of futures studies in management," *Futures*, vol. 40, no. 3, pp. 237–246, Apr. 2008.
- [3] D. Reis and R. Lobo, "Technological forecasting: the methodology used by a federation of industries in Brazil," *Australian Journal of Basic and Applied Sciences*, vol. 9, no. 20, pp. 503–509, 2015.
- [4] A. L. Porter *et al.*, "Technology futures analysis: Toward integration of the field and new methods," *Technological Forecasting and Social Change*, vol. 71, no. 3, pp. 287–303, Mar. 2004.
- [5] C. Cagnin, A. Havas, and O. Saritas, "Future-oriented technology analysis: Its potential to address disruptive transformations," *Technological Forecasting and Social Change*, vol. 80, no. 3, pp. 379–385, 2013.
- [6] R. SBRAGIA, "Apresentação do XXI Simpósio de Gestão da Inovação Tecnológica," *São Paulo: USP/FEA*, 2000.
- [7] F. Betz, "Strategic technology management," 1993.
- [8] A. Bernardes, G. G. Cummings, C. S. Gabriel, Y. D. M. Évora, V. G. Maziero, and G. Coleman-Miller, "Implementation of a participatory management model: analysis from a political perspective," *Journal of Nursing Management*, vol. 23, no. 7, pp. 888–897, 2015.
- [9] S. Valtolina, B. R. Barricelli, and Y. Dittrich, "Participatory knowledge-management design: A semiotic approach," *Journal of Visual Languages & Computing*, vol. 23, no. 2, pp. 103–115, Apr. 2012.
- [10] M. Godet, "Caixa de Ferramentas da prospectiva tecnológica," *Centro de Estudos de Prospectiva e Estratégia-CEPES, Lisboa*, 2000.
- [11] A. Eerola and I. Miles, "Methods and tools contributing to FTA: A knowledge-based perspective," *Futures*, vol. 43, no. 3, pp. 265–278, Apr. 2011.
- [12] W. Bell and J. K. Olick, "An epistemology for the futures field: Problems and possibilities of prediction," *Futures*, vol. 21, no. 2, pp. 115–135, Apr. 1989.
- [13] M. Jaspers, H. Banthien, and J. Mayer-Ries, "New forms of knowledge management in participatory foresight: The case of 'Futur,'" in *Eu-us seminar: New technology foresight, forecasting & assessment methods (Seville)*, 2004.

Complex Networks Analysis for Software Architecture: a Case Study on Hibernate

Daniel Henrique Mourão Falci, Bruno Rafael de Oliveira Rodrigues
Orlando Abreu Gomes and Fernando Silva Parreiras
Laboratory for Advanced Information Systems - LAIS
FUMEC University
Belo Horizonte, Brazil 30130-009
<http://www.fumec.br/lais>

I. INTRODUCTION

In the Software Engineering field, one has been observing an unceasing search for quantitative measures capable of assessing internal software attributes such as maintainability, reusability, agility, among others. The argument is that these characteristics, also known as architecturally significant requirements (ARS), when combined, determine the product quality [1], what in turn affects the development costs, particularly while under the software maintenance cycle.

A natural choice to study the association level among software components is through the usage of call graphs, where vertexes may represent software elements in a system, and the edges map their calls, giving shape to a complex network of relationships [2]. Such representation may be dynamic, indicating that it was captured during system execution, or static that on the contrary, analyzes the source code structure. Call graphs have proven its utility in many software development activities such as compiler optimization and program understanding [3].

Considering this scenario, the following research question emerge: "Which software attributes may be revealed through the application of common complex network analysis measures on call graphs?". We also intend to analyze the topological and basal properties of software in network theory field. In this work, we utilize the Hibernate library, a well-known Object/Relational Mapping (ORM) framework, widely employed in Java-based enterprise applications.

The rest of this paper is organized as follows: In section 2 we expose our methods and materials. In section 3 we discuss our results, and finally, in section 4 we conclude our work.

II. MATERIALS AND METHODS

This paper relied on the static call graph representation extracted from Hibernate library in its version 5.1.3¹. All the undertaken analysis presented here utilized the software Gephi² and the complex network analysis package named NetworkX³, for the Python language. We used the Gephi to

create the network's visual representations and to acquire its basal properties. Through of NetworkX library was performed data manipulations and the investigations in this study.

A. Call graph extraction

To extract the static call graph structure from Hibernate, we developed a tool named Call Graph Extractor (CGE⁴). This tool can read the bytecode of Java classes embedded into JavaArchive files (JAR files) with the goal of extracting the caller and the callee for all instructions of each class contained in the referred file. In other words, our software analyzes internal methods of a class, regardless of their visibility (public, private, static, and so on), creating a relationship table in an output file.

B. Graph modeling

In our method, for each extracted call, we create three vertexes: The caller method, the callee class, and the callee method. If the caller and callee classes are different, indicating an object transition, we link them with two edges observing the rule: $[(callerMethod, calleeClass), (calleeClass, calleeMethod)]$. If the caller and callee classes are equals, denoting an internal call, then the rule applied is different: $[(callerMethod, calleeClass), (callerMethod, calleeMethod)]$. It is worth noting that our representation cannot be considered bipartite due to the presence of edges linking same type vertexes, in our case, methods.

In order to constrain the graph representation to the library domain, we discarded the calls whose caller or callee instructions did not belong to the "org.hibernate" package, consequently removing any call to the native Java library. Following these procedures, we obtained a graph comprised of 27,556 vertexes and 57,919 edges.

III. RESULTS AND DISCUSSION

In this section we present the basal properties of Hibernate network analyzing its small-world, scale-free and community properties. Among the values for some of the main complex network measures are: average shortest path = 19.664; the average clustering coefficient = 0.194; the average shortest path

DOI reference number: 10.18293/SEKE2019-035

¹<http://www.hibernate.org>

²<https://gephi.org/users/download/>

³<https://networkx.github.io>

⁴<https://github.com/dfalci/callgraphextractor>

= 19.644; average degree = 2.02; network diameter = 62; modularity = 0.838 and the number of communities = 446.

In order to verify the existence of small world which is characterized by being a regular networks that can be highly clustered, like regular lattices, have small characteristic path lengths, like random graphs [4]. In our Hibernate network representation, we calculated the link creation probability p as shown by the equation 1, where l is the number of edges and n represents the number of vertexes. With the p value, we proceeded to the generation of a random network based the Erdős-Rényi model [5]. For the random network, we registered an average clustering coefficient $C = 0$ and an average shortest path $d = 3.479$. These values were obtained from the mean of the values obtained by five random networks. The value of C is derived from the very low value of p , which may produce random graphs with sparse connectivity, so sparse that may result in a disconnected graph, as it is the case. Through the comparison of the values obtained from Hibernate and random networks, reported in the equation 2. One may say that the Hibernate network presents small-world characteristics. Its clustering coefficient is much higher than the one of the random network, while the average shortest path in both networks is close, although the average shortest path obtained for Hibernate network is more than five times greater than the value achieved by the random network.

$$p = \frac{2l}{n(n-1)} = \frac{2(57,919)}{27,556(27,555)} = 0.0001525 \quad (1)$$

$$C_{Hibernate} \gg C_{Random} \quad \text{and} \quad d_{Hibernate} \approx d_{Random} \\ 0,194 \gg 0 \quad \text{and} \quad 19.664 \approx 3.479 \quad (2)$$

The Hibernate network degree distribution analysis suggests that it may be classified as a scale-free network. Scale-free indicates that the development of large networks is governed by robust self-organizing phenomena that go beyond the particulars of the individual systems [6]. It follows a power law distribution ($P(k) \sim k^{-\alpha}$) whose main characteristic is the existence of a few number of vertexes possessing a large number of connections while the vast majority of vertexes have a small number of links. The log-log plot reveal the angular coefficient $\alpha \approx 2.6$. One of the main properties of scale-free networks is its failure resistance. This definition though, must not be used in a software context, since a failure on the smallest software component may compromise the whole system. This finding is in compliance with previous studies [7], [8], [9].

The modularity analysis was based on the algorithm described by [10], known as Louvain method. The modularity value ($M = 0.838$) is close to the maximum possible value (I) which reveals a high network propensity to form communities. In fact, 446 communities were detected by this method, which results in an average value of 61.78 vertexes per class. The ten biggest classes, though, holds 47.45% of the total number of vertexes.

Rank	Betweenness	Degree	PageRank
1	org.hibernate.type.TypeFactory	org.hibernate.internal.CoreMessageLogger	org.hibernate.internal.util.StringHelper
2	org.hibernate.boot.cfgxml.spi.LoadedConfig	org.hibernate.internal.CoreMessageLogger.Slogger	org.hibernate.internal.CoreMessageLogger
3	org.hibernate.boot.MetadataSources	org.hibernate.engine.spi.SessionImplementor	org.hibernate.internal.CoreLogging

Figure 1. Top ranked software components

The Figure 1 shows the top 3 vertexes obtained from the analysis of the main complex network analysis measures when applied to Hibernate's call graph. Due to the lack of comparison ground for the task of describing the relevance of such software components, particularly over the Hibernate source code structure, we limited our analysis to present the top ranked software elements for each network measure.

IV. CONCLUSION

In this paper, we analyzed the properties of a widely employed Java-based software through its static call graphs. We investigated its topology that revealed a small-world and scale-free network, in compliance with the findings of [7], [9], [8]. Furthermore, we have identified that the network exhibits a strong propensity to form communities. Finally, we highlighted the top 3 software components for some of the main complex network analysis measures.

The tools developed in this work may benefit research in this field of study as it provides accessible means to create static call graphs for Java-based software. In future we plan to compare the traditional measures applied in software engineering domain with the ones obtained through the application of complex network analysis measures. We also plan to investigate the application of modularity as way of finding higher level software components and its respective relevance within system.

REFERENCES

- [1] L. Chen, M. A. Babar, and B. Nuseibeh, "Characterizing architecturally significant requirements," *IEEE software*, vol. 30, no. 2, pp. 38–45, 2013.
- [2] M. Pezzè and M. Young, *Teste e análise de software: processos, princípios e técnicas*. Bookman Editora, 2009.
- [3] J. Bohnet and J. Döllner, "Visual exploration of function call graphs for feature location in complex software systems," in *Proceedings of the 2006 ACM symposium on Software visualization*. ACM, 2006, pp. 95–104.
- [4] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.
- [5] P. Erdős and A. Rényi, "On random graphs i," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.
- [6] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.
- [7] S. Valverde and R. V. Solé, "Hierarchical small worlds in software architecture," *arXiv preprint cond-mat/0307278*, 2003.
- [8] L. Wang, Z. Wang, C. Yang, L. Zhang, and Q. Ye, "Linux kernels as complex networks: A novel method to study evolution," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009, pp. 41–50.
- [9] L. Ying and D.-w. Ding, "Topology structure and centrality in a java source code," in *Granular Computing (GrC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 787–789.
- [10] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

An Approach for Collecting Real Estate Development News

I. THE APPROACH

Fig. 1. The architecture of the proposed approach

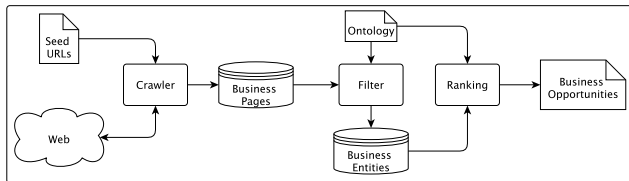


Figure 1 presents the architecture of our approach to collect, classify and rank real estate development news from the Web. We highlight that the crawler component of our approach firstly uses a set of seed URLs to collect real estate development news, storing them in a business pages corpus, re-feeding itself from URLs contained in the collected pages. The crawling flow used by our approach is similar to a previous work reported in [1].

Second, the filter component extracts textual features, i.e., page title, page description, and page text, from the real estate development news corpus. For this, it filters valid pages by removing pages from invalid URLs, duplicated, empty, redirected and private pages. Then, it performs stop words removal, stemming, and removal of punctuation, accents and special characters in order to increase the quality of the textual features. Finally, it exploits a real estate development ontology to extract entities from the textual features.

Third, the ranking component uses supervised algorithms to learn a ranking model in order to provide an ordered list with the most relevant business opportunities extracted from the Web.

II. PRELIMINARY EXPERIMENTS AND RESULTS

To evaluate our approach, we run experiments to answer the following research questions: i) how effective is our approach to collect and filter real estate development news? ii) which textual features provide better filtering performance? The evaluation of the entity recognition and ranking strategies will be carried out in future work.

Particularly, we use a dataset composed of 419 real estate development news, reported in Table I, previously collected from the Web and labeled by experts to evaluate two different algorithms used to generate the filtering models: SVM (Support Vector Machine) with linear kernel and RF (Random Forest). Additionally, we use nine configurations for training and test sets, varying the training and test percentages from 90-10 to 10-90, we performed 5-fold cross-validation [2],

and we report effectiveness in terms of accuracy, i.e., the percentage of true positives for all positive predictions. Moreover, we evaluate four different sources of textual features extracted from the business pages: i) Title (TO); ii) Title (TD) + Description; iii) Title + Full Text (TF); iv) Title + Description + Full Text (ALL).

TABLE I

NUMBER OF REAL ESTATE DEVELOPMENT NEWS COLLECTED USING OUR APPROACH.

Processing Steps	# Remaining Pages
Crawler	1,493
Duplicate Removal	1,034
Invalid URLs Removal	564
Empty Removal	558
Redirected Removal	466
Private Removal	419

Table II shows the accuracy of each learning algorithm used to filter real estate development news with different features for each training and test configuration schema. From Table II we observe that the title-only feature is less effective than the others, since it contains few words that are related to the business context. Additionally, we observe that SVM mostly outperforms RF with accuracy from 92% to 100% depending on the number of instances used in training.

TABLE II

FILTERING REAL ESTATE DEVELOPMENT NEWS ACCURACY.

Config.	RF				SVM			
	TO	TD	TF	ALL	TO	TD	TF	ALL
90/10	0,70	1,00	0,90	0,93	0,80	0,96	0,96	1,00
80/20	0,65	0,96	0,83	0,90	0,73	0,96	0,98	0,98
70/30	0,64	0,90	0,93	0,86	0,71	0,89	0,97	0,92
60/40	0,68	0,81	0,88	0,95	0,73	0,91	0,97	0,91
50/50	0,71	0,86	0,87	0,90	0,75	0,95	0,87	0,93
40/60	0,70	0,82	0,77	0,83	0,73	0,87	0,86	0,92
30/70	0,72	0,72	0,71	0,88	0,71	0,86	0,93	0,91
20/80	0,71	0,52	0,72	0,92	0,72	0,75	0,90	0,90
10/90	0,72	0,71	0,71	0,78	0,73	0,79	0,87	0,92

Recalling our first and second research questions, these observations attest the effectiveness of our approach to collect and filter real estate development news. In addition, we show that our textual features provide impact positively in the filtering performance.

REFERENCES

- [1] F. Hamborg, N. Meuschke, C. Breiteringer, and B. Gipp, "news-please: A generic news crawler and extractor," in *Proceedings of the 15th International Symposium of Information Science*, 2017.
- [2] R. Jain, *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. Wiley-Interscience, New York, 1991.

GraphQL Servers generation from R2RML mappings with morph-GraphQL (DEMO)

1st Ahmad Alobaid

Department of Artificial Intelligence
Universidad Politécnica de Madrid
Madrid, Spain
aalobaid@fi.upm.es

2nd Freddy Priyatna

Department of Artificial Intelligence
Universidad Politécnica de Madrid
Madrid, Spain
fpriyatna@fi.upm.es

3rd David Chaves-Fraga

Department of Artificial Intelligence
Universidad Politécnica de Madrid
Madrid, Spain
dchaves@fi.upm.es

4th Oscar Corcho

Department of Artificial Intelligence
Universidad Politécnica de Madrid
Madrid, Spain
ocorcho@fi.upm.es

Abstract—The adoption of GraphQL is on the rise, many companies and institutes are adopting it due to its ease of use, ease of maintenance, and hide the complexity from the user. Such advantages come from using a unified global schema and mapping it to the underlying data sources. The semantic web community has already adopted a similar way to map different data resources (e.g., R2RML). We present a novel way of generating GraphQL server from R2RML mappings.

Index Terms—GraphQL, R2RML, Wrapper, Adapter

I. INTRODUCTION

Facebook developed GraphQL¹ as an alternative to REST and made it open source to be used by the public in 2015. A GraphQL server consists of multiple components: schema, resolvers, and data sources. The GraphQL schema is the exposed view - an interface that the user (the person who writes queries) can use to access the underlying data sources. GraphQL resolvers are written codes (in a programming language) to link fields in the data sources to the exposed schema. The data sources are the where the data are stored and can be retrieved from such as a Relation Database (e.g., MySQL).

R2RML [1], published in 2012 by the RDB2RDF W3C Working Group, is a W3C recommendations for transforming the content of relational databases into RDF datasets. It allows the users to specify rules of how this transformation being done, such as how the URIs be generated, or which columns to be used in the transformation rules.

II. MORPH-GRAPHQL

In [2] we introduce morph-GraphQL², that takes as its input R2RML mappings and generates the corresponding GraphQL server. GraphQL engine interprets queries written in GraphQL and use the corresponding resolvers to fetch the data from the data sources. This workflow is shown in Fig. 1. R2RML

will be the input to morph-GraphQL and it will output the corresponding GraphQL resolvers and schema. The schema, resolvers and the data source are the input to the GraphQL engine.

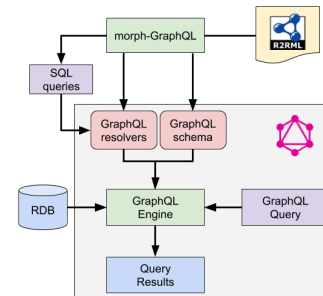


Fig. 1. Workflow of morph-GraphQL

In this demo we will show how we create R2RML mappings for the CSV files³ containing Star Wars data that is used as the example in the reference implementation⁴. Then we will use morph-GraphQL to generate GraphQL schema and resolvers from created mappings. Finally, we will evaluate some queries corresponding to the Star Wars example.

Acknowledgement: This work is supported by the Spanish Ministerio de Economía, Industria y Competitividad and EU FEDER funds under the DATOS 4.0: RETOS Y SOLUCIONES - UPM Spanish national project (TIN2016-78011-C4-4-R)

REFERENCES

- [1] S. Das, S. Sundara, and R. Cyganiak, "R2RML: RDB to RDF Mapping Language," <https://www.w3.org/TR/r2rml/>, accessed: 2018-12-07.
- [2] F. Priyatna, D. Chaves-Fraga, A. Alobaid, and O. Corcho, "morph-GraphQL: GraphQL resolvers generation from R2RML mappings." in *SEKE*, 2019.

¹<https://graphql.org/>

²<https://github.com/oeg-upm/morph-graphql>, deployed at <http://graphql.morph.oeg-upm.net>

³<https://github.com/oeg-upm/morph-graphql/tree/master/examples/starwars>

⁴<https://github.com/graphql/graphql-js>

Author Index

Abid, Saad Bin	355
Abreu Gomes, Orlando	774
Acher, Mathieu	541
Acuna, Silvia T.	479
Aguiar, Rui	503
Ait Oubelli, Lynda	748
Ait-Ameur, Yamine	748
Akram, Junaid	349
Alencar, Paulo	20, 415, 489
Alexandre, Tielle	233
Alkharabsheh, Khalid	361
Almeida, Hyggo	26, 82, 165, 171
Alobaid, Ahmad	291, 777
Alvarez Rodríguez, Jose María	64
Aman Shah, Akber	349
Amraoui, Hend	681
An, Dongdong	153
Anbang, Guo	713
Angele, Kevin	32
Anvik, John	205
Arcangeli, Jean-Paul	473
Ardelean, Alexandru	449
Assunção, Joaquim	513, 729
Aung, Moe Nandi	159
Ayachi Ghannouchi, Sonia	459
Azouzi, Sameh	459
Barbosa, Carlos Eduardo	770
Barbosa, Simone	325
Batista, Marcelo	135
Bedouet, Judicaël	748
Bența, Kuderna-Iulian	449
Bertuol, Gelson	661
Beserra, Leandro	215
Biffi, Stefan	693
Bixin, Li	612
Blersch, Martin	32
Blot, Elliott	227
Bordini, Rafael	71
Borges, Olimar	453
Bracco, Luciano	509
Brahmi, Zaki	459
Brito E Abreu, Fernando	199
Brito, Beatriz	579

Bruegge, Bernd	735
Bruel, Jean-Michel	473
Brunotte, Wasja	94, 245
C, Marimuthu	553
Cai, Xiaoshu	281
Cai, Ziyi	431
Cairo, Aloisio	199
Campos, Ursula	325
Cao, Junming	5, 410
Cao, Kaibo	425
Cao, Kunlin	535
Cardoso Borda Castro, Diego	770
Cardoso Brandão, Wladmir	776
Carneiro, Glauco	199, 497
Casanova Pietroboni, Carlos Antonio	509
Castro, John W.	479
Catania, Samantha	117
Ceretta Nunes, Raul	253
Cesar, Marcos	26
Chang, Haoming	701
Chausserie-Laprée, Benoît	748
Chaves-Fraga, David	291, 777
Chen, Chunqi	518
Chen, Cong	563
Chen, Dan	634
Chen, Junhua	141
Chen, Wenwang	634
Chen, Wenzhi	404
Chen, Xiang	425
Chen, Xing	367
Chen, Yangyang	563
Chen, Yimei	43
Chen, Yiyang	367
Chen, Yuanyu	634
Chen, Yucheng	535
Chen, Yuting	5, 307, 410, 518
Cheng, Wen	287
Cheng, Yaru	630
Chi, Xiaoxiao	11
Chimalakonda, Sridhar	493
Choma, Joelma	88
Chondamrongkul, Nacha	187
Coelho, Roberta	215
Colomé, Marcelo	253
Conte, Tayana	325
Corcho, Oscar	291, 777
Correia, Filipe Figueiredo	88

Costa, Alexandre	26, 82
Costa, Luis Felipe Coimbra	770
Cotos, José M.	361
Couto, Julia	39, 71, 453
Cowan, Don	20, 415
Craske, Antoine	687
Crespo, Yania	361
Cuenca, Javier	606
Curry, Edward	606
da Cruz Mello, Otávio	600
da Silva, Lucas Pereira	129
Da Silva, Tiago Silva	88
Daneva, Maya	398
Dantas, Emanuel	82
de Assis, Thiago Botti	55
De Battista, Anabella	509
de Carvalho, Cleuves	221
de Lara, Juan	479
de Lima Silva, Luis Alvaro	253
de Oliveira Rodrigues, Bruno Rafael	583, 774
de Oliveira Salim, Matheus	776
de Souza França, Renata	583
Deng, Fei	15, 102
Devanla, Gurudev	209
Diosan, Laura	547
Do, Canh Minh	107
Doborean, Dragos	547
Domingues Regateiro, Diogo	503
Dormuth, Jacob	717
Dorneles Soares, Heder	233
Du, Bowen	331
Du, Tianjiao	5
E. M. Almeida, Paulo	640
El-Fallah Seghrouchni, Amal	233
Elloumi, Mourad	681
Endo, André Takeshi	55
Engelmann, Debora	71
Exman, Iaakov	61, 75
Famá, Fernanda	221
Fan, Guisheng	319
Fan, Hongfei	331
Faria, João	646
Fei, Yuan	147, 265
Feng, Shihao	723
Feng, Wenlong	1

Fernandes, Paulo	513, 729
Fernandes, Sergio	589
Fernandez-Delgado, Manuel	361
Fernández-Izquierdo, Alba	573
Ferreira Salgado, Vinícius	776
Fonsêca, Carlos	766
Fontoura, Adriano	239
Fontoura, Lisandra	239
Fu, Duankang	307
Fu, Huiyuan	343
Fu, Lirong	404
Gabriel, Vagner	71
Galappaththi, Akalanka	205
Gao, Feng	15, 102
Gao, Jianhua	141
Gao, Min	523
García-Castro, Raúl	573
Garrido, Filipe	579
Gelman, Ben	717
Giallonardo, Ester	301
Gong, Xufang	469
Gorgonio, Kyller	165
Gorgônio, Kyller	221
Grechanik, Mark	209
Gu, Ming	707
Gu, Qing	425
Gu, Zuxing	707
Guerra, Eduardo	88
Guo, Junxia	43
Guo, Yirou	529
He, Xin	379
Heinz, Marcel	541
Hong, Zhong	563
Hu, Chuangshumin	367
Hu, Haibo	523
Hu, Jun	49
Huang, Mengxing	1
Huang, Qiguo	425
Huang, Zhiming	367
Huang, Zijie	141
Hübner Brondani, Camila	600, 661
Ikramov, Rustam	741
Iqbal, Nayyar	523
Ivanov, Vladimir	741

Jabeen, Farzana	529
Jabeen, Gul	349
Jain, Shivani	313
Jebali, Adel	113
Jemai, Abderrazak	113
Jiang, Jian-Min	563
Jiang, Siyu	431
Jin, Menglei	624
Jin, Xin	419
Johanssen, Jan Ole	735
K, Chandrasekaran	553
Kabir, Ahmedul	655
Karre, Sai Anirudh	618
Kathrein, Lukas	693
Kleebaum, Anja	735
Klinder, Jil	94
Kortum, Fabian	94
Kou, Huaizhen	11
Koussaifi, Maroun	473
Lai, Chih-Ju	557
Lakkundi, Chaitanya S.	493
Larrinaga, Felix	606
Larzul, Béatrice	748
Le Borgne, Alexandre	465
Li, Bixin	443, 469
Li, Chi	707
Li, Lefeng	760
Li, Menglong	1
Li, Qingshan	287
Li, Tengfei	153
Li, Wei	5
Li, Xiaojie	535, 723
Li, Xin	437
Li, Yinghua	379
Li, Yingling	49
Li, Zengyang	385
Li, Zheng	43
Liang, Peng	398
Lin, Lan	193
Liu, Hanwen	11
Liu, Jing	153
Liu, Peiyu	404
Liu, Qin	331
Liu, Weibin	624, 630
Liu, Xiumin	650
Liu, Zhanghui	367

Liu, Zheng	650
Llorens, Juan	64
Longo, Douglas Hiura	129
Loparo, Kenneth	281
Lopes, Adriana	325
Lopes, Lucelene	513
Lu, Gang	259, 265
Lu, Jiawei	275
Lu, Lu	431
Lu, Peng	557
Lucas, Edson	489
Lucio, Levi	355
Lulu, Wang	612
Luo, Chao	535
Luo, Yi	331
Ly, Jiancheng	723
Lämmel, Ralf	541
Lüder, Arndt	693
M. Souza, Cinthia	640
Maciel, Alexandre	766
Magalhaes, Ana Patricia	589
Mahajan, Vishal	355
Manzoni Fontoura, Lisandra	600, 661
Mao, Xiaoguang	713
Marczak, Sabrina	453
Marques, Nuno C.	497
Masyagin, Sergey	741
Mathur, Neeraj	618
Matos, Ecivaldo	579
Mei, Shanshan	529
Meixner, Kristof	693
Melo dos Santos, Glaucia	20
Menegassi, André Augusto	55
Meneguzzi, Felipe	39
Meng, Zhangyuan	410
Mhamdi, Faouzi	681
Micallef, Mark	117, 594
Mo, Shaocong	419
Monteiro, Miguel	497
Moore, Jessica	717
Morayo, Adedjouma	391
Moreno, Valentín	64
Mortágua Pereira, Óscar	503
Mou, Zeya	701
Mourão Falci, Daniel Henrique	774, 776
Neuhaus, Priscilla	39

Ni, Chao	425
Ni, Zeyu	410
Niu, Nan	713
Nogueira, Ana Filipa	687
Nunes, João	165
Nunes, Nuno	579
Ogata, Kazuhiro	107, 159, 181
Oliveira, Toacy	20, 415, 489
Ouyang, Liubo	337
Paech, Barbara	735, 754
Palisetti, Sanjana	553
Pan, Haibo	275
Pantoja, Carlos	233
Pedro, Antonio	26
Pedroza, Gabriel	391
Peng, Rong	567
Pereyra Rausch, Fernando	509
Perez, Quentin	465
Perkusich, Angelo	26, 82, 165, 171, 221
Perkusich, Mirko	26, 82, 165, 171
Persch, Henrique	239
Phyo, Yati	159
Ping, Luo	349
Poggi, Francesco	301
Porter, Chris	117, 594
Prikladnicki, Rafael	453
Priyatna, Freddy	291, 777
Qi, Jiyang	385
Qi, Lianyong	11
Qiang, Yin	612
Qiong, Zeng	563
Qiu, Shaojian	431
Qu, Yili	419
R. G. Meireles, Magali	640
Ramos, Felipe	26
Raza, Mushtaq	646
Relvas, Antonio	497
Ren, Ranci	479
Resende, Antonio	199
Ribeiro, José Carlos	687
Rocha, Mauricio	135
Rosa, Jean	579
Rossi, Davide	301
Rottoli, Giovanni Daian	509

Ruiz, Duncan	39
Ruiz, Duncan D.	453
Sabou, Marta	693
Sadiq, Ali Zafar	655
Saha, Anju	313
Sakib, Kazi	655
Salva, Sébastien	227
Sang, Jun	523
Santos, Danilo	221
Santos, Glaucia	415
Saraiva, Renata	171
Sassi, Salma	113
Schab, Esteban	509
Schneider, Kurt	94, 245
Seiler, Marcus	754
Sergeant, Emilien	687
Shakirov, Ruslan	741
Shen, Beijun	5, 307, 410, 518
Sheng, Feng	259
Shi, Canghong	723
Shu, Hongping	563
Shuai, Jia	419
Silote Neto, Florindo	583
Silva Parreiras, Fernando	583, 774, 776
Silva, Luiz	165
Silva, Thiago	485
Simão, Adenilso	135
Sinderen, Marten	398
Sirazidtinov, Ilyas	741
Slater, David	717
Song, Qi	535
Song, Tianyou	331
Sousa, Thiago	135
Souza, Jano	770
Spanier, Assaf	61
Strüber, Daniel	245
Su, Jianmin	437
Succi, Giancarlo	741
Sun, Dongzhen	265
Sun, Haiying	153
Sun, Jing	187, 297
Sun, Meng	271
Sun, Tao	177
Sun, Weidi	271
Sun, Xin	193
Sun, Yingcheng	281

Taboada, Jose Angel	361
Takada, Shingo	123
Tang, Haoran	379
Tang, Hui	337
Tang, Xiangru	385
Tao, Linmi	529
Thapaliya, Ananga	741
Tichy, Walter	32
Tormasov, Alexander	741
Trouilhet, Sylvie	473
Trætteberg, Hallvard	485
Tudor, Luke	297
Urtado, Christelle	465
Vale, Anderson	589
Valente, Pedro	579
Vauttier, Sylvain	465
Venigalla, Akhila Sri Manasa	493
Vieira, Renata	71
Viertel, Fabien Patrick	245
Vilain, Patrícia	129
Vincent, Jean-Marc	729
Vinicius, Marcus	82
Vital, Rachel	20
Viterbo, Jose	233
Vuong, Thi Anh Tuyet	123
Wallach, Harel	75
Waltersdorfer, Laura	693
Wang, Bangchao	567
Wang, Chong	398
Wang, Dongdong	443
Wang, Dongjing	373
Wang, Hai H.	297
Wang, Junjie	49
Wang, Lu	287
Wang, Qi	177
Wang, Qing	49
Wang, Shangwen	713
Wang, Shanshan	760
Wang, Tao	398
Wang, Tong	443, 469
Wang, Xiao Jie	343
Wang, Zhihao	385
Wang, Zhuo	567
Wang, Zonghui	404
Warren, Ian	187

Wehrmann, Jonatas	39
Wei, Bingyang	297
Weigelt, Sebastian	32
Wen, Wushao	437
Winckler, Marco	485
Winkler, Dietmar	693
Wu, Jiecheng	707
Wu, Linbo	15
Wu, Qinyue	5
Wu, Ruobiao	147
Wu, Xi	535
Wu, Xianyu	723
Xiang, Chen	667, 675
Xiang, Hong	523
Xiao, Gang	275
Xiao, Guangyi	337
Xiao, Lili	265
Xie, Zefeng	419
Xin, Wang	535
Xing, Weiwei	624, 630
Xiong, Yunxiang	518
Xu, Jun	275
Xu, Kaihui	373
Xu, Qiwen	147
Y, Raghu Reddy	618
Yan, Yunqiang	15, 102
Yang, Cheng-Zen	557
Yang, Guanzhong	701
Yang, Kang	319
Yang, Xiaoxing	437
Yang, Xingguang	319
Yang, Yixiao	667, 675
Yang, Zongyuan	259
Yao, Wenbin	343
Yi, Xin	713
Yin, Jiaqi	147, 259, 265
Yin, Jing	723
Yin, Youbing	535
You, Zhi-Jun	557
Yu, Chang Wu	331
Yu, Dongjin	373
Yu, Huiqun	319
Zaina, Luciana	88
Zeng, Jun	379
Zenha-Rela, Mário	687

Zhang, He	287
Zhang, Shi	563
Zhang, Shiyu	529
Zhang, Yelian	469
Zhang, Yu	1
Zhao, Ruilian	43
Zhao, Yaxin	567
Zheng, Wenbo	419
Zheng, Zengwei	634
Zhou, Huan	275
Zhou, Jiliu	535
Zhou, Min	707
Zhou, Shufan	307
Zhu, Hongming	331
Zhu, Huibiao	147, 259, 265
Zimeo, Eugenio	301
Ziviani, Fabrício	583
Zufarova, Oydinoy	741

Additional Reviewers

Afsharchi, Mohsen
Albuquerque, Danyllo
Allian, Ana Paula
Ayora, Clara

Barat, Souvik
Barletta, Vita Santa
Braga Gomes,

Campos, Eduardo
Cassano, Fabio
Castro, John W.
Chen, Zheyi
Cordeiro, André
Costa, Alexandre
Cézane, Dalton

Deval, Vipin
Dilorenzo, Ednaldo
Dixit, Abhishek
Domingues Regateiro, Diogo
Dray, Gerard
Dwivedi, Vimal
Dósea, Marcos

Fekri, Mohammad Navid
Feng, Yuzhou
Fernandes Gomes Da Silva, Rodrigo
Ferreira, Juan M.
Ferreira, Thiago
Filho, Emanuel

Galappaththi, Akalanka
Gamage, Dimuthu
García, Ignacio
Georgieva, Petia
Ghosh, Ananda
Ghosh, Aritra
Guy, Ed

Kane, Shridhar
Khan, Mujahid
Khoshgoftaar, Taghi
Kormiltsyn, Aleksandr

Lambolais, Thomas
Legretto, Alessandra
Li, Yi
Lichtenthäler, Robin
Lin, Yun
Liu, Ai
Lu, Yuteng

Magues, Daniel
MagÜes, Daniel
Malaviya, Sugandha
Manner, Johannes
Mihret, Zelalem
Moraga, Ma Ángeles
Moreira, José
Morgan, Jameson
Mottu, Jean-Marie

Nebut, Clémentine
Neto, Ademar
Nunes, João

Park, Sumin
Pereira, Luiz
Perez, Quentin
Perkusich, Mirko
Prado Lima, Jackson
Pérez-Castillo, Ricardo

Ramos, Felipe
Rocha, Adriano
Rodríguez, Francy
Roychoudhury, Suman
Rybarczyk, Ryan

Sehovac, Ljubisa
Silva, Raissa
Sobreira, Victor
Song, Jiyoung
Sunkle, Sagar

Tchechmedjiev, Andon
Teixeira, Lucas
Theis Gerald, Ricardo
Tian, Yifang
Tibermacine, Chouki

Udokwu, Chibuzor

Vilar, Rodrigo A.

Wen, Junye

Winzinger, Stefan



SEKE

**Lisbon, Portugal
July 10 - 12, 2019**

**Proceedings of the 31st
International Conference on
Software Engineering &
Knowledge Engineering**

Copyright © 2019
Printed by
KSI Research Inc. &
Knowledge Systems Institute
156 Park Square
Pittsburgh, PA 15238 USA
Tel: +1-412-606-5022
Fax: +1-847-679-3166
Email: seke@ksiresearch.org
Printed in USA, 2019
ISBN 1-891706-48-9 (paper)
ISSN 2325-9000 (print)